Report No. 457

A HARDWARE REALIZATION OF A DECOMPOSITION ALGORITHM

by

RICHARD DEAN GARTON

June 1971

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 457

A HARDWARE REALIZATION OF A DECOMPOSITION ALGORITHM

by

RICHARD DEAN GARTON

June 1971

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

* This report was supported in part by the Department of Computer Science and the Coordinated Science Laboratory, and submitted to the Graduate College in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. Gernot Metze for his many helpful discussion periods and his continual encouragement during the course of this project. The use of the IBM 360 computer was made available by the Digital Computer Laboratory. Materials for the preparation of the thesis were supplied by the Coordinated Science Laboratory. The typing was performed by Mrs. Rose Lane and Mrs. Sandy Bowles.

The author is grateful to his mother and father who were largely responsible for the financial aspects of his undergraduate education. Finally, the author expresses appreciation to his wife, Barbara, who has shown a great deal of patience and who was of considerable help in preparing the rough draft.

TABLE OF CONTENTS

Page

1. INTRODUCTION 1

2. CURTIS' ALGORITHM 3

2.1. Simple Decompositions 3
2.2. Complex Decompositions 9
2.3. Improper Decompositions 11
2.4. The Final Algorithm 15

3. HARDWARE FOR EFFECTIVE REALIZATION OF ALGORITHM 18

3.1. Where Hardware Can Help 18
3.2. Minterm Rearranger 22
3.3. Column Multiplicity Recognizer 35

4. CONCLUSIONS 42

LIST OF REFERENCES 45

APPENDIX

A. COMPUTER PROGRAMS AND DISCUSSION OF RESULTS FOR MINTERM REARRANGER SHIFTS AND CYCLES OF SHIFTS 46

B. COMPUTER PROGRAM FOR DERIVING MINTERM ORDERS FROM VARIABLE ORDERS 81

LIST OF TABLES

Table Page

1. Decomposition charts needed for execution of Algorithm..... 19

2. Comparison of maximum cycle size with total number of permutations..... 34

3. Rough cost estimate of proposed hardware to investigate a function of n variables..... 44

LIST OF FIGURES

Figure		Page
1.	Example decomposition chart	4
2.	Construction of a simple disjunctive decomposition	6
3.	Construction of a simple nondisjunctive decomposition	8
4.	Construction of an improper multiple decomposition	13
5.	Flow chart of the final algorithm	17
6.	Minterm values on a decomposition chart. (a) Manual chart (b) Machine chart	21
7.	Block diagram of the proposed system	23
8.	Correspondence between L/B/F notation and minterm order	24
9.	Implementation of a shift to obtain minterm order shown in Figure 8	26
10.	Two-leveled tree	29
11.	Variable order rearranger	32
12.	Block diagram of minterm rearranger	36
13.	Column multiplicity recognizer for -/2/1 charts	37
14.	Circuitry to change length of column	39
15.	Circuitry to change overlapping variable size	41
16.	Program for -/3/1(4) Charts	49
17.	Program for -/2/2(6) Charts	58
18.	Program for 1/2/1(12) Charts	68
19.	Program to find minterm orders for cost 3, -/3/1(4) library	82

1. INTRODUCTION

The problem of realizing Boolean functions of a large number of variables has proven to be quite difficult. Manual application of Karnaugh maps and the Quine-McCluskey method become awkward for functions of more than 5 or 6 variables. The methods may be programmed and extended to functions of more than 6 variables, but the fact that only two-level realizations can be obtained is a severe limitation. In 1957 Ashenhurst (1) presented a unified decomposition theory to deal with this problem. Decomposing a function means breaking the function up into subfunctions which may be realized independently and then combined to form the desired function. This immediately implies that realizations may involve more than two levels. The increased flexibility results in the discovery of cheaper realizations. Also, because the original function is expressed in terms of subfunctions, it will effectively have a lower number of variables and be easier to work with.

Several attempts have been made to develop programs and algorithms based on Ashenhurst's theory. These include computer programs developed by Karp, McFarlin, Roth, and Wilts (2), by Schneider and Dietmeyer (3), and by Shen and McKellan (4); and algorithms developed by Curtis (5), and by Karp (6). This is not meant to be an exhaustive list. Other methods have been developed using the decomposition concept, but not using Ashenhurst's theory. For example, Marin (7) discusses a method based on solving systems of Boolean equations. The advantage is that realizations other than the standard AND-OR-NOT type are easily found. Unfortunately, only two-level realizations can be obtained. Another

example is the multi-leveled realization method developed by Davidson (8) based on NAND functions. However, the method is not usable for other types of functions.

The method proposed here is based on the decomposition algorithm advanced by Curtis (5) from Ashenhurst's theory. While automation of the algorithm was thought possible by its author, originally, only manual applications were presented. The algorithm is reviewed and two fundamental tasks which may be implemented much more efficiently by hardware than by programming are discovered. Suggested hardware necessary to realize these tasks is discussed in detail.

2. CURTIS' ALGORITHM

2.1. Simple Decompositions

A function has a simple disjunctive decomposition if it may be written in the form

$$f(A,B) = F(g(A),B),$$

where A and B are disjoint subsets of the total set of variables $x_1, x_2, \ldots x_n$. The variables in A are called the bound variables; those in B are called the free variables. If A and B were to have members in common, the decomposition would be known as a simple nondisjunctive decomposition. To avoid confusion, the variables common to A and B will be placed in a third subset, C. The simple nondisjunctive decomposition may then be written in the form

$$f(A,B,C) = F(g(A,C),B,C).$$

The variables in C are called the overlapping variables. This will be the notation in general, always to write disjoint subsets of the total set of variables.

Simple decompositions are discovered with the aid of a decomposition chart. A decomposition chart is simply a listing, in the form of a matrix, of the 1's and 0's that define the function. The values the bound variables may have define the columns of the matrix, and the values the free variables may have define the rows. See Figure 1 for an example. The construction of a decomposition chart is similar to the construction of a Karnaugh map except that decomposition charts can

A = {3 bound variables}

B = {2 free variables}	000	001	010	011	100	101	110	111
00								
01								
10								
11								

Figure 1. Example decomposition chart.

vary in shape corresponding to different numbers of members in the bound and free sets. Also, while the construction of one map is sufficient for the investigation of a function when using Karnaugh maps, many charts must be constructed to investigate the decompositions of a function. A new chart is needed for each different choice of bound and free variables.

The column multiplicity of a decomposition chart is the number of different types of columns the chart has. A simple disjunctive decomposition exists if the column multiplicity of the associated decomposition chart is no greater than two. A proof of this may be found in Curtis (5).

The construction of a simple disjunctive decomposition is based on the following expansion

$$F(g(A),B) = F(0,B)\ \bar{g}(A)\ v\ F(1,B)\ g(A).$$

One of the distinct columns on the chart is chosen to be F(0,B), the other F(1,B). Then g(A) is formed by replacing a column by 0 if it is the same as F(0,B) and by 1 if it is the same as F(1,B). F(g,B) is formed by placing F(0,B) and F(1,B) side by side. An example has been worked in Figure 2. The importance of having no more than two different columns can be seen. There is no way to incorporate a third column.

Notice that the choice of which column is to be F(0,B) and which is to be F(1,B) is arbitrary. Either choice being perfectly valid leads to the conclusion that there are two possible realizations once it has been determined that this type of decomposition exists.

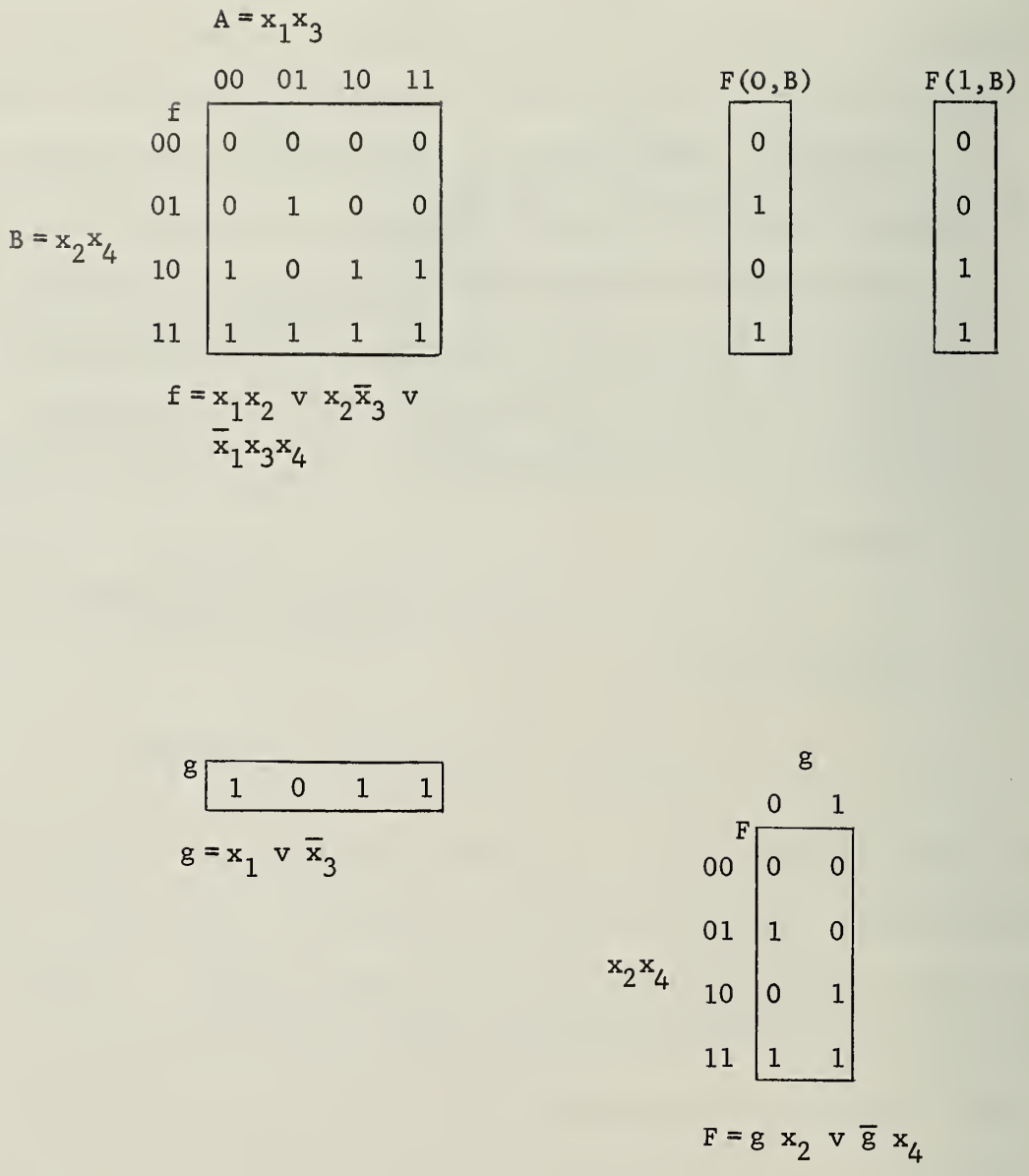


Figure 2. Construction of a simple disjunctive decomposition.

The nondisjunctive case with one overlapping variable is treated by dividing the chart in half vertically. The overlapping variable will then be 0 on the left hand side of the chart and 1 on the right hand side of the chart. A nondisjunctive decomposition exists if the right hand side and the left hand side each exhibit a disjunctive decomposition (column multiplicity no greater than two).

The construction of a nondisjunctive decomposition is shown in Figure 3. The individual disjunctive decompositions on each side of the chart are constructed as indicated above. The resulting F_0 and F_1 are placed side by side to form F, and the resulting g_0 and g_1 are placed side by side to form g.

Once it has been determined that a nondisjunctive decomposition exists, there are four ways of realizing it. This is because the disjunctive decomposition on the right side may be realized in two ways and the disjunctive decomposition on the left may be realized in two ways. Together they may be paired up in four ways.

The extension to a set of overlapping variables with more than one member is obvious. The chart is divided into four parts for two overlapping variables, eight parts for three overlapping variables, and, in general, into 2^n parts for n overlapping variables. The number of ways of realizing the decomposition increases as 2^{n+1}.

$C = x_5$

$x_5 = 0$ $\quad$ $x_5 = 1$

$A = x_2x_4$ $\quad$ $A = x_2x_4$

g_0 $\quad$ g_1

F_0	0	1
0	0	1
1	0	0
2	0	1
3	1	0

$B = x_1x_3$

f	00	01	10	11	00	01	10	11
00	0	1	1	0	0	0	0	0
01	0	0	0	0	1	1	1	1
10	0	1	1	0	0	1	1	1
11	1	0	0	1	1	1	1	1

F_1	0	1
0	0	0
1	1	1
2	0	1
3	1	1

$$f = x_1x_2x_5 \vee x_1x_4x_5 \vee x_2\bar{x}_3\bar{x}_4\bar{x}_5 \vee \bar{x}_2\bar{x}_3x_4\bar{x}_5 \vee x_1\bar{x}_2x_3\bar{x}_4 \vee x_1x_2x_3x_4 \vee x_3x_5$$

g	g_0				g_1			
	0	1	1	0	0	1	1	1

$$g = x_2\bar{x}_4 \vee x_4x_5 \vee \bar{x}_2x_4\bar{x}_5$$

x_5g

F	00	00	10	11
00	0	1	0	0
01	0	0	1	1
10	0	1	0	1
11	1	0	1	1

x_1x_3

$$F = gx_1x_5 \vee g\bar{x}_3\bar{x}_5 \vee \bar{g}x_1x_3 \vee x_3x_5$$

Figure 3. Construction of a simple nondisjunctive decomposition.

2.2. Complex Decompositions

A complex decomposition is one that has more than one subfunction. Examples of complex decompositions are the iterative decomposition:

$$f(A,B,C) = G(g(h(A),B),C),$$

and the multiple decomposition:

$$f(A,B,C) = F(g(A),h(B),C).$$

Either of these may be extended to include more than two subfunctions. In general, a complex decomposition will be a combination of these two types.

The amazing property of complex decompositions is that their existence may be predicted from a knowledge of all the simple decompositions. It has been proven in Curtis (5) that by applying the following two rules, the complete set of complex decompositions can be obtained.

If these two simple decompositions exist:
$f(A,B,C) = G(g(A,B),C)$
$f(A,B,C) = H(h(A),B,C),$
then this complex decomposition will exist:
$f(A,B,C) = G(k(h(A),B),C).$

If these two simple decompositions exist:
$f(A,B,C,D) = G(g(A,B),C,D)$
$f(A,B,C,D) = H(h(A,C),B,D),$
then this complex decomposition will exist:
$f(A,B,C,D) = K(k(m(A),n(B),p(C)),D)$

where C and/or D may be empty.

These rules may be generalized: If another simple decomposition fitting into the general pattern can be added to the list, another subfunction will appear in the complex decomposition obtainable.

A complex decomposition may be constructed by constructing several simple decompositions and combining the results. The construction steps of the complex decompositions of each of the theorems are shown below.

Step 1: $f(A,B,C) = G(g(A,B),C)$
Step 2: $g(A,B) = k(h(A),B)$
Result: $f(A,B,C) = G(k(h(A),B),C)$

Step 1: $f(A,B,C,D) = K(q(A,B,C),D$
Step 2: $q(A,B,C) = r(m(A),B,C)$
Step 3: $r(m,B,C) = s(m,n(B),C)$
Step 4: $s(m,n,C) = k(m,n,p(C))$
Result: $f(A,B,C,D) = K(k(m(A),n(B),p(C)),D)$

Each step requires the construction of only a simple decomposition, but the combined effort results in the construction of a complex decomposition.

Complex nondisjunctive decompositions are also possible. They are based on the following theorem (or a generalization of this theorem if more than two decompositions can be included):

If these simple nondisjunctive decompositions exist:
$f(A,B,C) = G(g(A,u),B,v)$
$f(A,B,C) = H(h(B,w),A,y)$,
then this complex nondisjunctive decomposition will exist:
$f(A,B,C) = K(g(A,u),h(B,w),v \cap y)$
where $u \cup v = w \cup y = C$

The proof of this theorem is in Curtis (5).

Complex decompositions are important because they will generally result in cheaper realizations than simple decompositions. The argument for this is the same as the argument originating the investigation of decompositions: Low cost realizations of a function can be obtained by first realizing a subfunction and then using this subfunction to help realize the desired function. Complex decompositions require first

realizing more than one subfunction and then using these subfunctions to help realize the desired function. Each new subfunction reduces the number of variables in the next function up. This hopefully reduces the cost of realizing that function and eventually the cost of realizing the desired function.

2.3. Improper Decompositions

An improper decomposition is a complex decomposition in which either the free set or the bound set is composed completely of overlapping variables. Directly from the two basic types of complex decompositions come the following two types of improper decompositions: the iterative improper decomposition

$$f(C,B) = F(g(h(C),B),C),$$

and the multiple improper decomposition:

$$f(C,B) = F(g(C),h(C),B).$$

By definition, these types are restricted to have two subfunctions only. Generalizations involving more than two subfunctions do exist, and will be discussed later in this section.

Each of the above types can be recognized by checking the appropriate decomposition charts. An iterative decomposition exists if a chart has a column multiplicity no greater than 6, and the 6 different columns are of the following types: the two trivial columns, all 1's and all 0's; any two nontrivial columns; and the complements of the two

nontrivial columns. A multiple decomposition exists if a chart has a column multiplicity no greater than 4. Proofs of the above criteria may be found in Curtis (5).

Recall that a simple decomposition exists if a chart has a column multiplicity no greater than 2. The column multiplicity of a chart is an important characteristic. The existence of simple disjunctive, simple nondisjunctive, complex, and improper multiple decompositions can all be recognized by knowing only the column multiplicity of a decomposition chart. Improper iterative decompositions, on the other hand, require in addition to knowing the column multiplicity, information about the content of the columns. Extracting this additional information will make investigating improper iterative decompositions much more time consuming than any of the other types. The problem becomes worse when the investigation is to be done by a machine. For this reason, improper iterative decompositions will be eliminated from further consideration. It is felt that this is not a serious deletion since a wealth of other choices still exists. The improper iterative decomposition as well as its generalization is discussed in Curtis (5).

The method of constructing an improper multiple decomposition is based on the following expansion:

$$
\begin{aligned}
F(g(C),h(C),B) = {} & F(0,0,B)\ \bar{g}(C)\ \bar{h}(C)\ v \\
& F(0,1,B)\ \bar{g}(C)\ h(C)\ v \\
& F(1,0,B)\ g(C)\ \bar{h}(C)\ v \\
& F(1,1,B)\ g(C)\ h(C).
\end{aligned}
$$

An example construction is shown in Figure 4. First the labels F(0,0,B), F(0,1,B), F(1,0,B), and F(1,1,B) are arbitrarily assigned to the four

	C = $x_1x_4x_5$								F(00B)	F(01B)	F(10B)	F(11B)
	f											
B = x_2x_3	0	0	1	0	0	0	0	0	0	0	0	1
	1	0	1	0	1	1	0	0	0	1	0	1
	1	0	1	0	1	1	1	1	0	1	1	1
	1	1	0	1	1	1	0	0	1	1	0	0

$$f=\bar{x}_1x_2x_3x_5 \vee x_1x_2\bar{x}_3 \vee x_1x_3\bar{x}_4 \vee x_3\bar{x}_4\bar{x}_5 \vee \bar{x}_1\bar{x}_2x_4\bar{x}_5 \vee x_2\bar{x}_3\bar{x}_5$$

		gh			
g	F				
0 0 1 0 0 0 1 1	x_2x_3	0	0	0	1
		0	1	0	1
		0	1	1	1
h		1	1	0	0
1 0 1 0 1 1 0 0					

$g=x_1x_4 \vee x_4\bar{x}_5$

$h=\bar{x}_1\bar{x}_5 \vee x_1\bar{x}_4$

$F=\bar{g}\bar{h}x_2x_3 \vee \bar{g}h(x_2 \vee x_3) \vee g\bar{h}x_2\bar{x}_3 \vee gh(\bar{x}_2 \vee \bar{x}_3)$

Figure 4. Construction of an improper multiple decomposition.

different columns found on the decomposition chart. Next, g(C) is constructed by replacing the columns identical to F(1,0,B) and F(1,1,B) with a 1, and the columns identical to F(0,0,B) and F(0,1,B) with a 0. Likewise, h(C) is constructed by replacing the columns identical to F(0,1,B) and F(1,1,B) with a 1, and the columns identical to F(1,0,B) and F(0,0,B) with a 0. Finally, F(g,h,B) is formed by placing F(0,0,B), F(0,1,B), F(1,0,B), and F(1,1,B) side by side. The construction is not unique. The arbitrariness in the assignment of the four columns allows $4! = 24$ different ways of doing it.

The improper multiple decomposition may be generalized to include more than two subfunctions. The result is called a generalized tree decomposition and is written

$$f(C,B) = F(g_1(C),g_2(C),\ldots g_k(C),B).$$

It has been shown by Curtis (9) that a generalized tree decomposition exists if the decomposition chart has no more than 2^k different columns. Some work has been done to develop an efficient design algorithm based on the generalized tree, for example in articles by Curtis (9), (10), and by Karp (6). The big advantage of this type of decomposition is that it always exists. If a decomposition chart has x different columns it can be realized by a generalized tree having 2^x subfunctions. This is nice for hand calculations, because every chart that is formed is guaranteed to give some results. It is still advisable to look at several charts, however, because some may give better results than others when the cost of realizing the decomposition is considered.

2.4. The Final Algorithm

Certain decompositions are called trivial decompositions because they always exist. They usually will not help in simplifying the function and will not be considered in the final algorithm for this reason. Simple disjunctive decompositions with bound set sizes of 0 or n variables are, for obvious reasons, two types of trivial decompositions. Another type is a simple disjunctive decomposition with a bound set size of 1 variable. The decomposition implied here would have a subfunction of only one variable. Since the subfunction would either be the variable or its complement, it is not very useful. Simple nondisjunctive decompositions which have bound set sizes of 0, 1, or n variables are also trivial because they involve considering trivial simple disjunctive decompositions. Improper multiple decompositions with overlapping set sizes of 0, 1, or n variables are trivial because they would again involve subfunctions containing none, one, or all of the variables. In addition, improper multiple decompositions with an overlapping set size of n-1 variables are trivial. The reason is that they always exist, as can be seen by the following identity:

$$f(x_1 \ldots x_n) = f(x_1 \ldots 1 \ldots x_n)\, x_i \vee f(x_1 \ldots 0 \ldots x_n)\, \bar{x}_i .$$

An improper multiple decomposition is always possible by choosing $f(x_1 \ldots 1 \ldots x_n)$ and $f(x_1 \ldots 0 \ldots x_n)$ as its two subfunctions. While these decompositions are not included as improper multiple decompositions, they are included as generalized tree decompositions.

A flow chart of Curtis' Algorithm is shown in Figure 5. The Algorithm is based on the following proposition: The decompositions of a function with the smallest number of members in the overlapping set will usually result in the cheapest realizations. Accordingly, the Algorithm is a loop which examines charts in order of increasing overlapping set size. At the end of each pass of the loop, if no decompositions were found, the overlapping set size of the charts for consideration during the next pass is made one greater. If decompositions were found, examination of any further charts is halted, and the best realization based on the decompositions found is chosen as probably being the cheapest. If all nontrivial decompositions are exhausted, the generalized tree decompositions are considered.

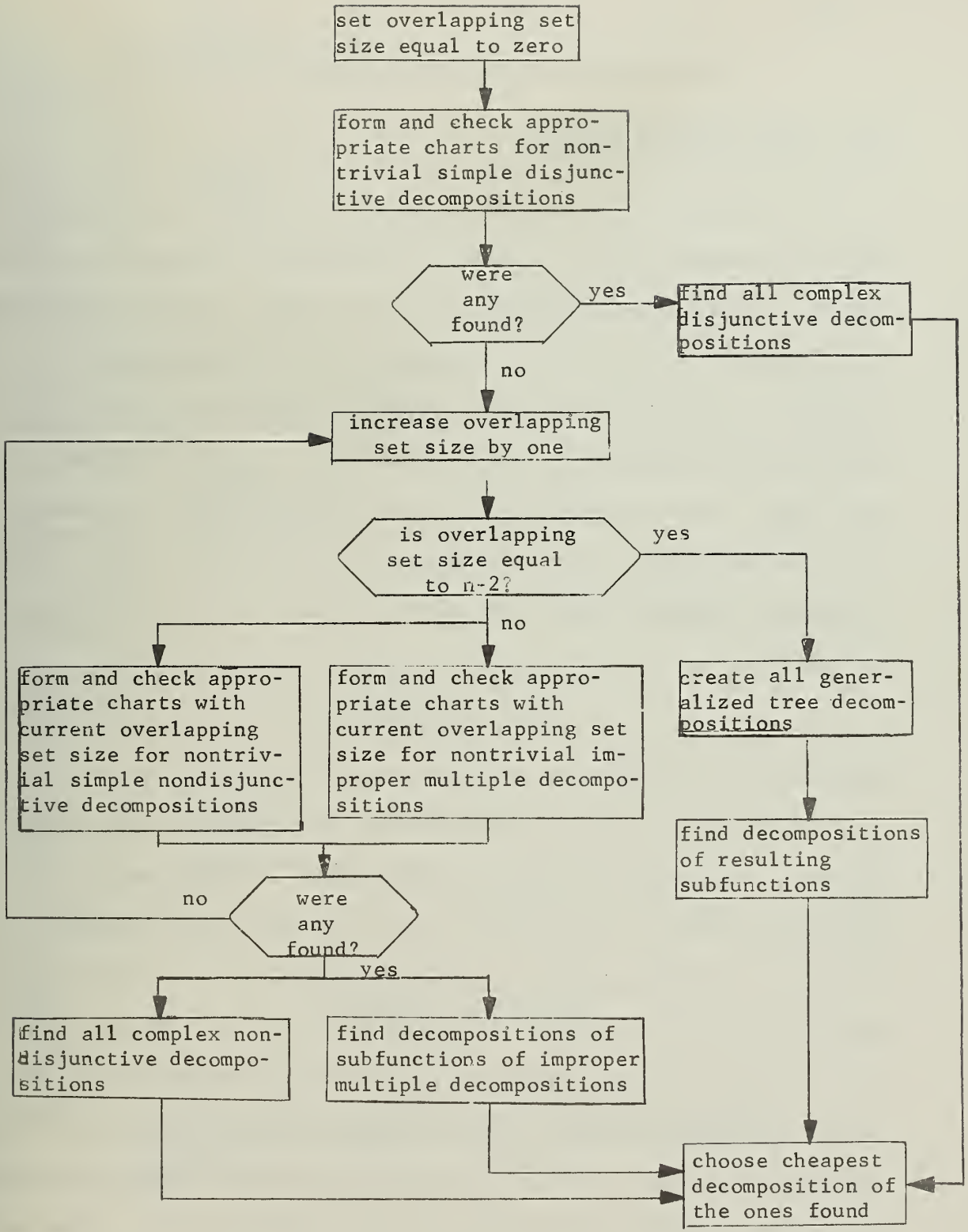


Figure 5. Flow chart of the final algorithm.

3. HARDWARE FOR EFFECTIVE REALIZATION OF ALGORITHM

3.1. Where Hardware Can Help

As can be seen from the flow chart in Figure 5, most of the work in executing Curtis' Algorithm is forming and checking decomposition charts. Exactly how many charts need to be formed and checked will now be calculated. A chart with L variables in the overlapping set, B variables in the bound set, and F variables in the free set will be represented by the notation L/B/F. A dash will be used to represent empty sets. The L variables for the overlapping set can be chosen in C(n,L) different ways. The B variables for the bound set can be chosen in C(n-L,B) different ways. The variables left will be in the free set. In all, there are

$$C(n,L) \cdot C(n-L,B)$$

different charts based on a fixed L,B, and F.

Table 1 lists all the nontrivial decomposition chart sizes which will be needed to execute the Algorithm. In parenthesis after the L/B/F notation is the number of charts of that size, found by evaluating the above equation. The -/B/F charts used for investigating simple disjunctive decompositions are actually the same as the L/-/F charts needed for investigating improper multiple decompositions, and need not be generated twice. The variables defining the columns of the charts will be bound variables for simple disjunctive decompositions and overlapping variables for improper multiple decompositions. These charts can also be used in investigating the generalized tree decompositions.

	C=0	C=1	C=2	C=3	C=4
n=3 simple disjunctive	-/2/1(3)				
n=4 simple disjunctive	-/3/1(4) -/2/2(6)				
improper multiple			2/-/2(6)		
simple nondisjunctive		1/2/1(12)			
n=5 simple disjunctive	-/4/1(5) -/3/2(10) -/2/3(10)				
improper multiple			2/-/3(10)	3/-/2(10)	
simple nondisjunctive		1/3/1(20) 1/2/2(30)	2/2/1(30)		
n=6 simple disjunctive	-/5/1(6) -/4/2(15) -/3/3(20) -/2/4(15)				
improper multiple			2/-/4(15)	3/-/3(20)	4/-/2(20)
simple nondisjunctive		1/4/1(30) 1/3/2(60) 1/2/3(60)	2/3/1(60) 2/2/2(90)	3/2/1(60)	

Notation: L/B/F(N) denotes that N decomposition charts with L overlapping, B bound, and F free variables must be examined.

Table 1. Decomposition charts needed for execution of Algorithm.

When manually forming and checking a chart, the geometry of the chart specifies the size of the columns. Some other way of denoting columns must be found for machine use. The most economical way is to store the length of the column and join the columns in one long list. The manual chart of Figure 6a is shown as a machine would represent it in Figure 6b. The positions have been labeled by minterm numbers. The convention in labeling minterms used here assumes that the subscripts of the variables are arranged in descending order from left to right. For example, minterm number 10 is $x_4\bar{x}_3x_2\bar{x}_1$. Since a minterm is either a 1 or a 0, each position of the machine chart is a single flip flop.

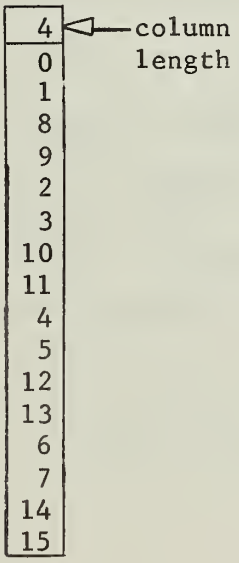
Creating the various charts would be an awkward task to program. Assume the starting condition of the chart is to list the minterms in order from 0 to 15 beginning at the top. Now assume it is desired to form the chart shown in Figure 6b. This amounts to a complicated rearrangement of the minterms. Each minterm would no doubt have to be moved separately into its new position.

A hardware approach can simplify the problem enormously. The complicated shift can just be wired in by hooking each flip flop output to the flip flop input of the position to which the minterm is to be moved.

Finding the column multiplicity of a chart is another task which can be done efficiently by a special piece of hardware. In Section 3.3 a design will be discussed which compares all columns in parallel to check for identity.

x_4x_1 \ x_3x_2	00	01	10	11
00	0	2	4	6
01	1	3	5	7
10	8	10	12	14
11	9	11	13	15

(a)



(b)

Figure 6. Minterm values on a decomposition chart. (a) Manual chart (b) Machine chart.

A hardware implementation of the entire algorithm would be impractical; the work remaining after the above two jobs are done is largely bookkeeping, which could best be done by a small computer. Small computers especially designed so external devices may be connected to them have recently become popular. These computers are known as minicomputers. It will be assumed that a minicomputer is available. It will be responsible for executing the algorithm in general. When it needs to form and check decomposition charts, it will turn to the external devices. The external devices consist of a minterm register, which holds the values of the minterms; a minterm rearranger, which shifts the minterm register to form the necessary decomposition charts; and a column multiplicity recognizer, which finds the number of different types of columns the chart has. A block diagram of the arrangement is shown in Figure 7. The hardware realizations of the external devices will be discussed in the next two sections. Interfacing between the minicomputer and the devices will be necessary, but will not be discussed here since it is dependent on which minicomputer is used.

3.2. Minterm Rearranger

The purpose of the minterm rearranger is to form the necessary decomposition charts by shifting the minterm register. A correspondence between the L/B/F notation of a chart and the minterm order of a chart can be established as follows. Refer to Figure 8. The positions of the minterm flip flops are numbered from 0 to 15 in binary beginning at the

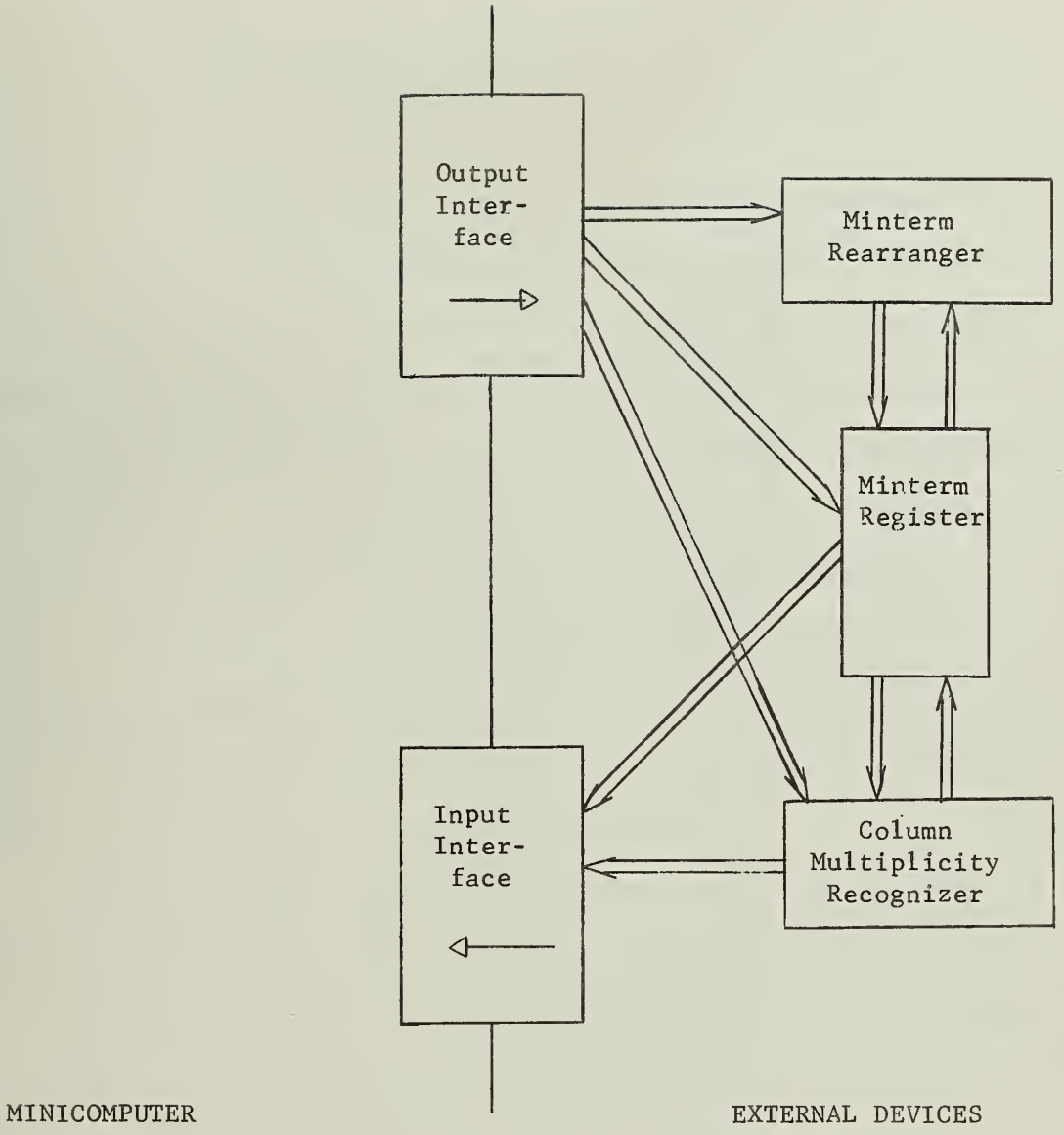


Figure 7. Block diagram of the proposed system.

minterm register	x_3x_2/x_4x_1 variable order 4 2 8 1 weights	minterm order
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	8
3	0 0 1 1	9
4	0 1 0 0	2
5	0 1 0 1	3
6	0 1 1 0	10
7	0 1 1 1	11
8	1 0 0 0	4
9	1 0 0 1	5
10	1 0 1 0	12
11	1 0 1 1	13
12	1 1 0 0	6
13	1 1 0 1	7
14	1 1 1 0	14
15	1 1 1 1	15

Figure 8. Correspondence between L/B/F notation and minterm order.

top. The L/B/F notation is placed above the binary numbers, one variable over each digit. The weights of the variables used in the minterm representation are used to weight the binary numbers corresponding to the positions of the minterm register in converting them to base ten. As can be verified by comparing Figure 8 and Figure 6b, the result is the order in which the minterms must appear. Ignoring the slashes, the L/B/F order will be called the variable order. Several other examples of deriving minterm orders from variable orders are given by the computer program in Appendix B. A simplification in the notation of a variable order is used by the program and will also be used here. Instead of writing x's with subscripts, only the subscripts will be written. For example, $x_3x_2x_4x_1$ becomes simply 3241. Since the minterm order can always be derived from the variable order, for the bulk of the investigations only the variable order need be considered.

Once the desired minterm order is known, wiring of the shift paths necessary to form this order is simple. The standardized starting order of listing the minterms in increasing magnitude from top to bottom will be assumed. To get the new order, each flipflop is connected by an AND gate to the input of the flip flop at the position to which the minterm is to be moved. Master-Slave flip flops are suggested to avoid race conditions. The shift is accomplished in one clock pulse. Figure 9 shows how a shift would be implemented to get the minterm order shown in Figure 8.

After the shift has been used once, there would be nothing wrong with enabling the AND gates and shifting again. Another machine

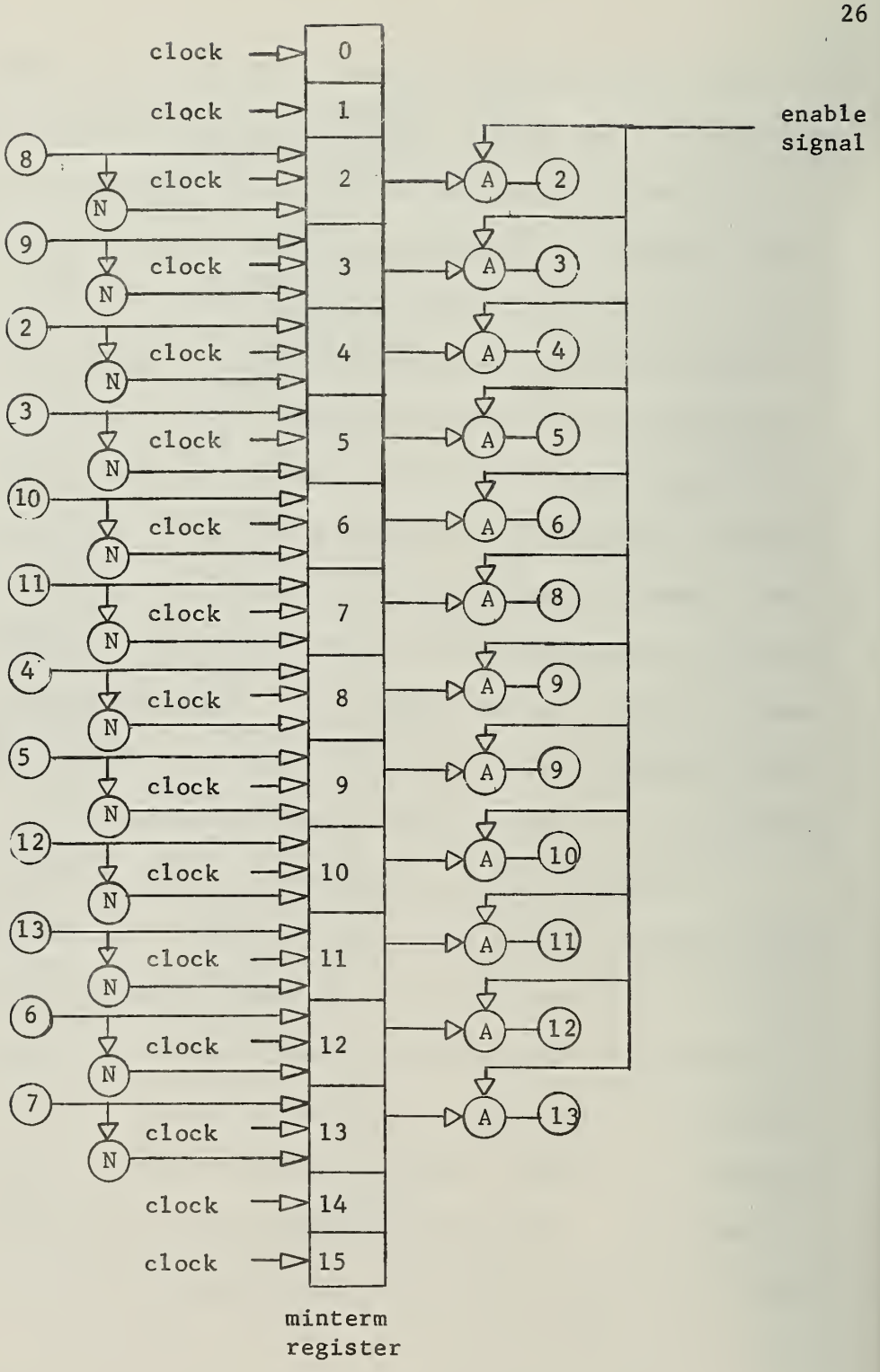


Figure 9. Implementation of a shift to obtain minterm order shown in Figure 8.

chart would be formed without the use of any additional hardware! If the procedure is continued, new machine charts will be generated until eventually a repetition occurs. Repeated use of the shift implemented in Figure 9 would result in the following cycle: 4321, 3241, 2431, 4321.

The fact that repeated applications of shifts gives rise to cycles shows the direction in which to proceed. Using the fewest number of different types of shifts, cycles must be found which will form the different classes of charts necessary to execute the Algorithm. The necessary charts have previously been listed in Table 1 of the last section. Consider the simplest example of the class of three charts of type -/2/1. The beginning variable order is the order 321. A shift of the type 321 to 213 can be applied three times to give the required three charts and restore the order to the original order The cycle is: 321, 213, 132, 321.

Assuming that the 321 to 213 shift has been implemented to form the charts of three variables, it can be used to help form charts of four variables. There is a cost involved, however. The shift must be converted from a three-variable shift, which involves 8 minterms, to a four-variable shift, which involves 16 minterms. The conversion is accomplished by wiring the second group of 8 flip flops the same way the first group of 8 was wired. The cost, while substantial, is still only half of what it would cost to implement an entirely new shift.

The extended shift alone obviously will not be sufficient to generate all of the required charts. At least one new shift must be implemented which will move the number 4 so it has a chance to get into different positions. Deciding which shifts to implement to get the

four variable charts is not going to be easy. As soon as more than one shift may be used, trying different subsets of shifts in different orders becomes necessary. It can easily be seen that the process of selection is beyond a trial and error procedure.

To investigate all possibilities, a computer program has been written. The program, results, and discussion of results are given in Appendix A. The method used by the program will be discussed here. The method is general and may be used for any number of variables. All possible cycles are generated based on a tree composed of nodes and branches connecting these nodes. Each node has a variable order associated with it. The beginning variable order is associated with the top node. Nodes of successive levels are found by applying all possible shifts to each of the nodes on the level above it. Cycles correspond to any chain of branches from the top node to any node on the bottom level. See Figure 10.

Fortunately, there are several constraints which tend to prune the tree and keep it from getting too large. The strongest constraint is that each of the charts in the class under consideration is to be generated only once. This implies that not only must identical charts not be repeated, but also equivalent charts must not be repeated. Equivalent charts are charts which contain the same members in the overlapping set, bound set, and free set. For example, the charts x_4x_3/x_2x_1, x_3x_4/x_2x_1, x_4x_3/x_1x_2, and x_3x_4/x_1x_2 are all equivalent to each other. Only one needs to be generated to recognize a decomposition

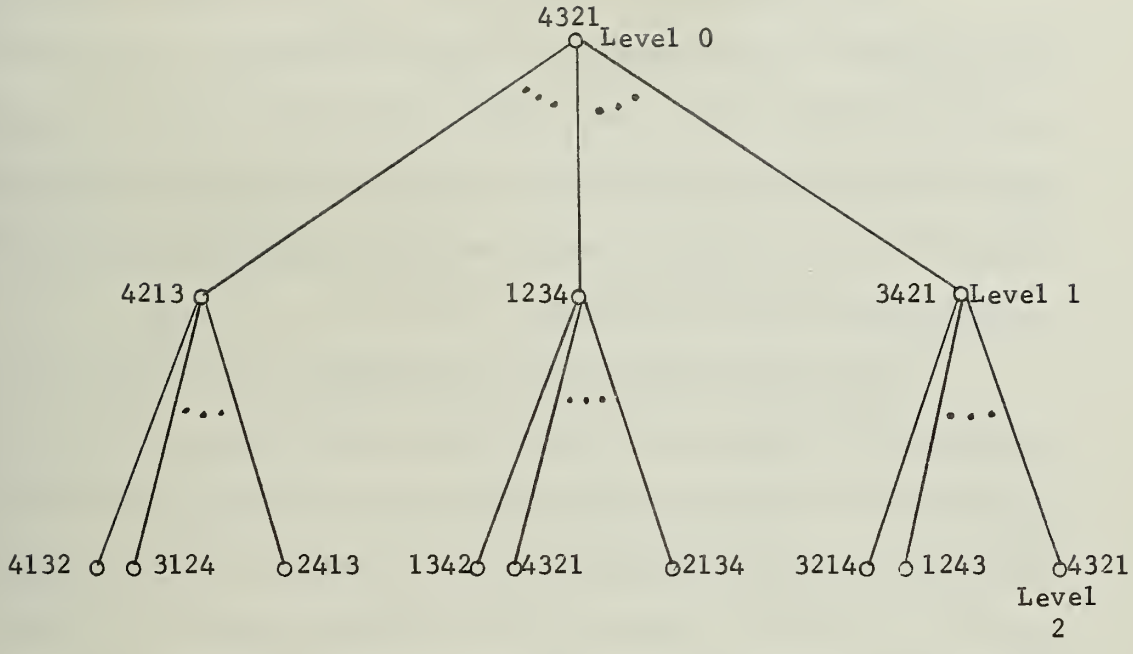


Figure 10. Two-leveled tree.

with bound set x_4x_3 and free set x_1x_2. In fact, generating any more than just one of them is a waste of time.

The "no repetition" constraint is implemented in the program by dividing all the permutations of four variables into partitions of equivalent charts. When a new shift is performed in advancing to the next level of the tree, the partition covered by the new variable order is found. If this partition has previously been covered, the sequence is abandoned; if not, it is kept.

A heuristic employed in the selection of the shifts to be considered also serves to limit the size of the tree. Since four numbers may be permuted in 24 possible ways, there are actually 24 different shifts that could be considered. Six of these shifts involve only the numbers 3, 2, and 1. These six shifts are really shifts of three variables, not four. It is known that a shift involving four variables will be necessary so that the 4 will have a chance to move around. It is also hoped that the number of new shifts to be added will be no more than one or two. For these reasons, shifts involving only three variables are not as likely to be used as shifts involving four variables, and have not been included. The three-variable shift actually chosen to form the three-variable charts will be included, however, since it can be extended to a four-variable shift at half the cost of implementing a true four-variable shift.

Cost is another one of the constraints. The sequence involving the addition of the fewest number of new shifts is the cheapest. Once the cost of a sequence reaches the maximum, whatever it is specified to

be, only those shifts already used are allowed to be used in forming successive levels. Sequences using any other shifts are deleted before they are even tried.

One last constraint is used in printing out the results. Only those sequences whose last variable order is the same as the starting variable order are printed. It would be very inconvenient if the cycle did not leave the minterms in the same order in which it found them. The next cycle, not starting at a standardized place, would not generate the same charts all of the time. The sequence generated would be a valid one, but it would be repeatable only if the starting order were the same. The easiest way to have repeatable results is to standardize the starting order by requiring a sequence to always restore the order to the order with which it began.

Based on the results of the programs, the hardware implementation of the necessary shifts can be carried out as indicated earlier. However, there is a certain amount of difficulty in keeping track of the variable order corresponding to the charts. The minterm rearranger shifts the minterms in the minterm register, but no record is kept of the corresponding change in variable order. There are two solutions to the problem of keeping track of the variable order. One is to work out ahead of time what the orders will be and place them in a table. The other is to build a variable order rearranger which would work along with the minterm rearranger. The design of a variable order rearranger is shown in Figure 11. The gate implemented is the 3241 gate of figure 9. The variable order rearranger amounts to no more than a minterm rearranger

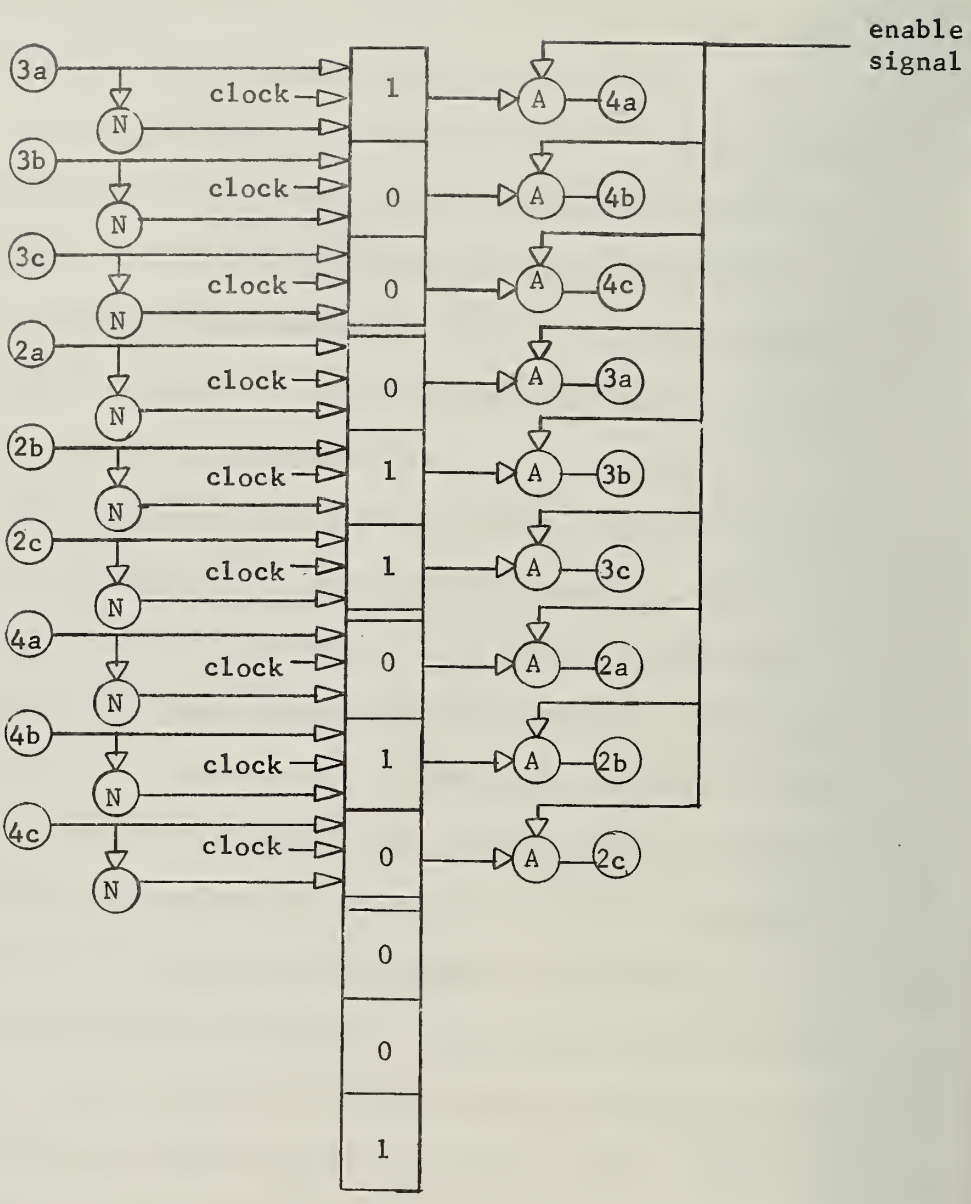


Figure 11. Variable order rearranger.

with three bits for each minterm. The numbers 1, 2, 3, and 4 are stored in the three bits and all three bits are shifted together.

A completely different approach to the whole problem is to generate all permutations of the variable order and select the ones that are needed. This method is attractive because it eliminates the need to find appropriate shifts and cycles. A method for generating all permutations which requires only one new shift for each variable added has been discovered by Johnson (11). Furthermore, there is a formula by which the variable order may be calculated directly from the chronological order. This eliminates the necessity of a table or variable order rearranger.

The main disadvantage of this second method is that many charts will be generated which will never be used. Time will be wasted in between generating the required charts while these useless charts are generated. How serious the problem is can be seen in Table 2, which compares the maximum number of charts any cycle of n variables has with the total number of permutations. As can be seen, the total number of permutations grows much faster than the maximum chart size. For 6 variables the ratio is 8:1. Also, use of this second method means cycles of all sizes will execute at the same speed, making the disparity for smaller cycles even greater. Until a thorough investigation of the first method is made and how many shifts must be added for each variable added is known, no adequate comparison can be made. It is thought that the first method will prove to be superior because of the time wasted by the second method.

n	maximum cycle size	number of permutations
3	3	6
4	12	24
5	30	120
6	90	720

Table 2. Comparison of maximum cycle size with total number of permutations.

For either method used, the hardware will be as shown in the block diagram of Figure 12. The only item not previously discussed in this diagram is the instruction register with decoder. The instruction register supplies the sequence of shifts to be performed. The sequences may either be supplied from the minicomputer, or exist as resident microcode.

3.3. Column Multiplicity Recognizer

The purpose of the column multiplicity recognizer is to interpret the minterm register as a decomposition chart and find its column multiplicity. The design presented here examines the columns of the chart from top to bottom and compares them in parallel through the use of a bus. Associated with each column is a flip flop whose state is changed if the column agrees with the column currently on the bus. The column multiplicity is obtained by counting the number of times different columns must be gated onto the bus until all of the flip flops associated with the columns have changed states.

The design for a chart of three variables with a column length of 2 is shown in Figure 13. Two flip flops of the minterm register, corresponding to one column of the chart, are connected to each control box. Each control box contains the master-slave flip flop to be associated with the column connected to it, circuitry to compare this column with the bus, circuitry to cause this column to be placed on the bus, and circuitry to communicate with the boxes above and below it.

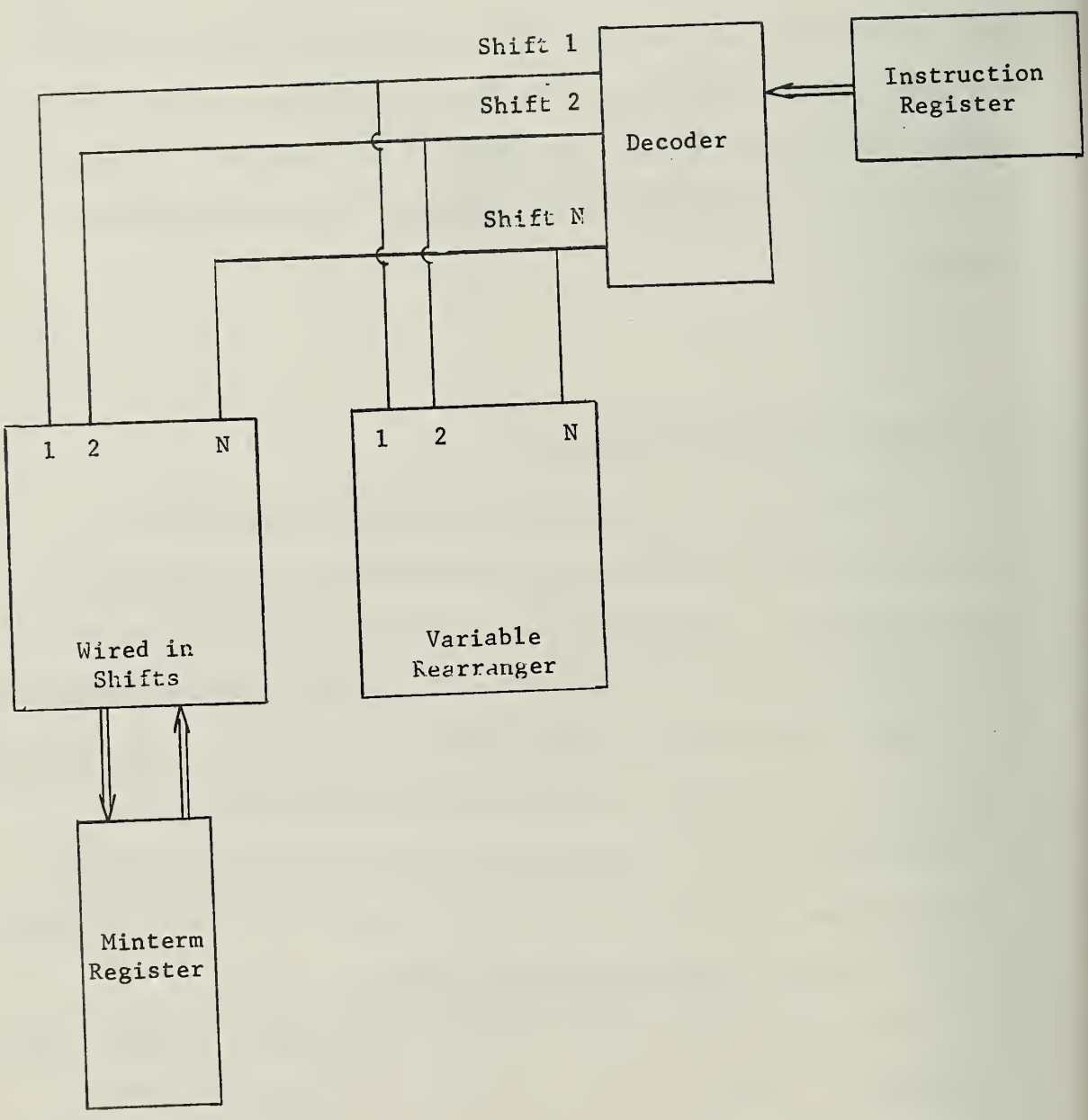


Figure 12. Block diagram of minterm rearranger.

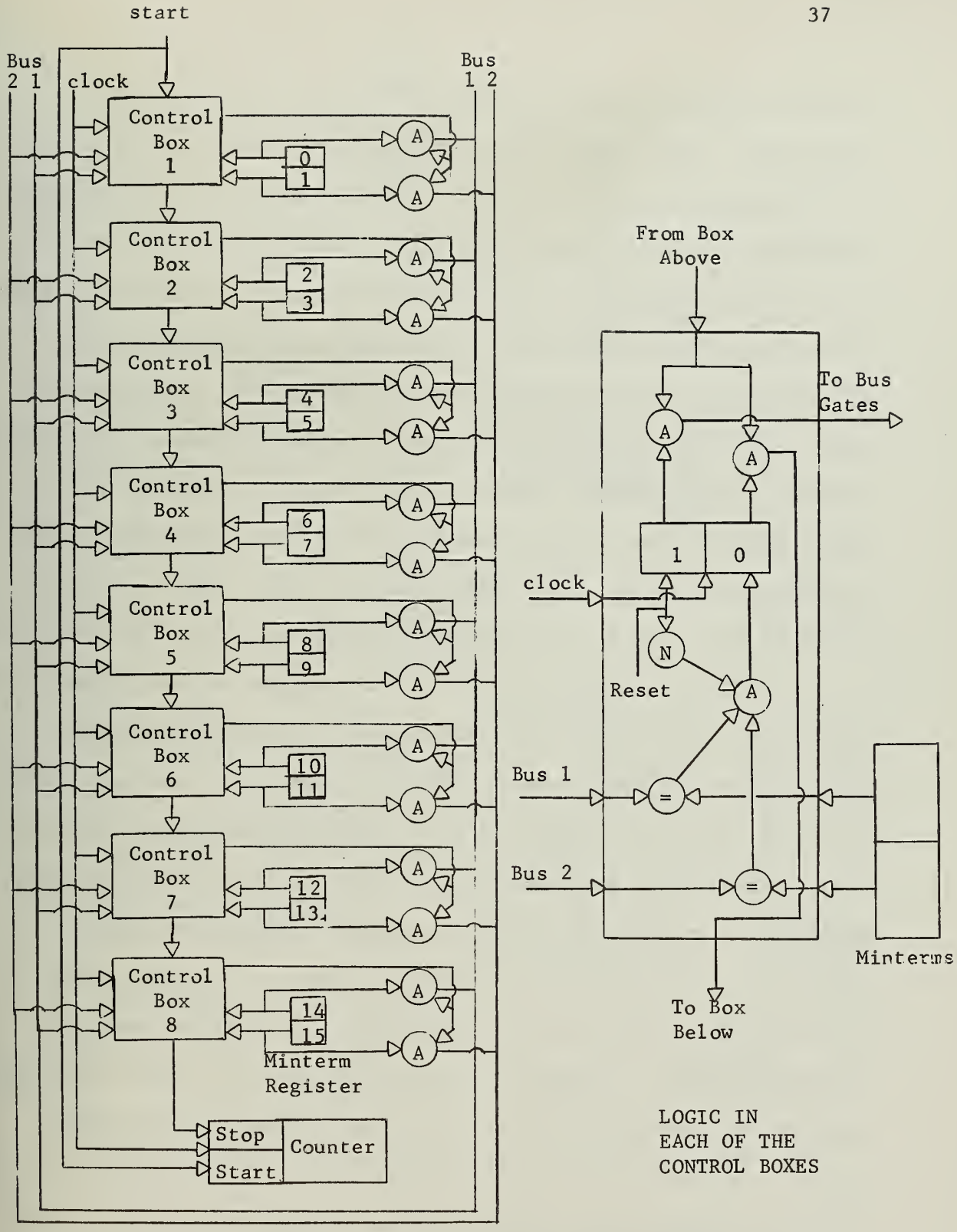


Figure 13. Column multiplicity recognizer for -/2/1 charts.

The flip flop inside each control box is reset at the beginning of the examination. The output signals from one box to the box below it and to the bus gates are then all zero. With the clock low, the start wire is energized. The flip flops of the minterm register associated with the top control box are then gated onto the bus. Each control box receives these signals and compares them to the signals from the flip flops of the minterm register which are associated with it. If the signals agree, the state of the flip flop inside the control box will be changed, when the clock goes high. With its flip flop in the changed state, the flip flops of the minterm register associated with the control box can never be gated onto the bus. Also, the signal into that control box from the box above will always be sent directly through the box to the box below.

When the clock goes low, the start signal will go through the top control box and down to the next control box which has not had the state of its flip flop changed. The flip flops of the minterm register associated with this control box will be gated onto the bus. The flip flop inside any box whose associated minterm register flip flops agree with the bus signals will be changed when the clock goes high again. The procedure continues until the start signal has rippled through all of the control boxes and out the other end. To count the number of different columns, a counter is started when the start wire is energized, advanced by one with each clock signal, and stopped when the start signal comes out of the last control box.

To change the length of column, more wires are added to the bus. Figure 14 shows how odd-even numbered control box pairs are to be

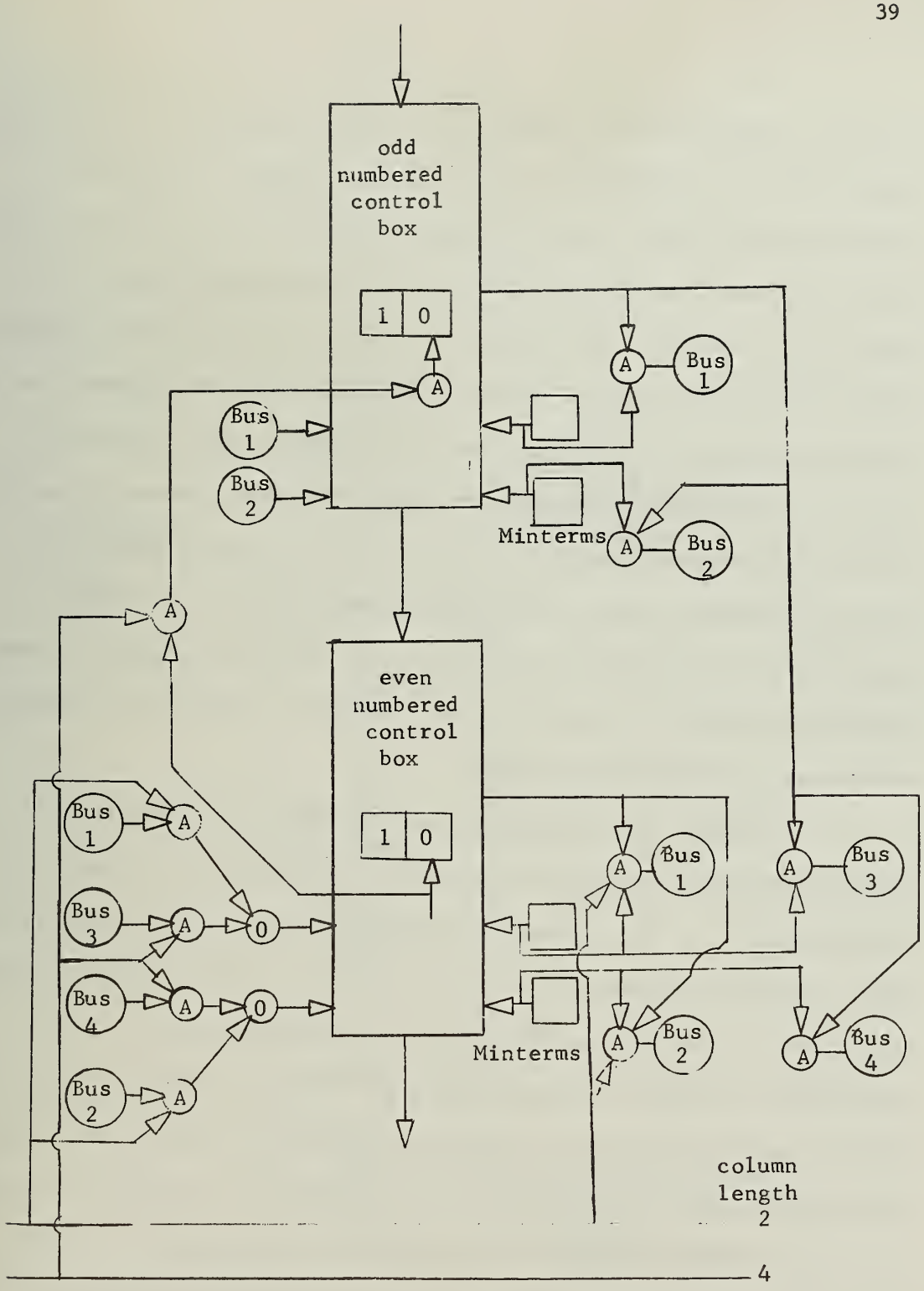


Figure 14. Circuitry to change length of column.

wired so a column length of either 2 or 4 can be selected. The column length of 4 is accomplished by using the gate signal from the odd numbered control box to gate the minterm register flip flops associated with it and also those associated with the even numbered control box below it onto the bus. The input to the flip flop in the even numbered control box is also made an input to the AND gate which produces the input to the flip flop in the odd numbered control box. This prevents the flip flop in the odd numbered control box from being changed unless the flip flop in the even numbered control box will also be changed. Columns of length 8 can be obtained by connecting units of length 4 together just as the units of length 2 were connected together to make columns of length 4. In general, this recursive method may be used to form columns of any desired length.

A scheme for dividing the chart in half for use with overlapping variables is shown in Figure 15. Two counters are needed, one for each half. If one overlapping variable is specified, both counters begin counting with the start signal and each is stopped when the signal ripples out of the last control box of its respective half. If 0 overlapping variables are specified, the last control box in the top half is connected through an AND gate to the first control box in the bottom half. When the examination is completed, the column multiplicity will be the value of the bottom counter; the top counter is ignored. More than 2 overlapping variables may be accommodated by extending the arrangement in the obvious manner.

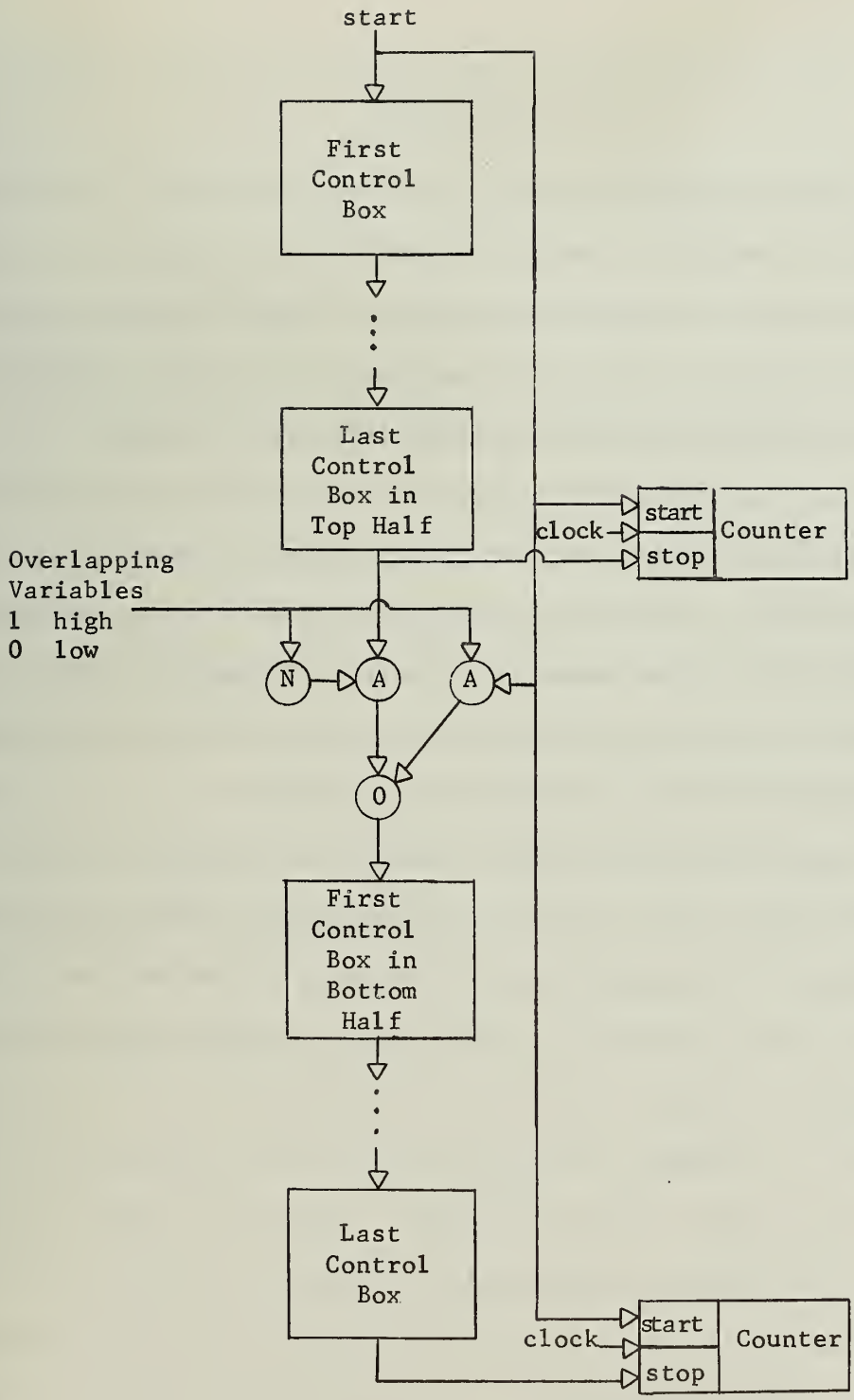


Figure 15. Circuitry to change overlapping variable size.

4. CONCLUSIONS

Hardware realizations for two fundamental tasks necessary to execute Curtis' Algorithm have been described. A question still remains as to whether only the desired decomposition charts should be created or whether decomposition charts corresponding to all variable orders should be created and the desired ones selected. The first solution to the problem is more efficient, but requires a complicated analysis. The analysis necessary for four variables has been given in the form of computer programs. The method can be continued for the investigation of a larger number of variables. Interfacing, dependent on which minicomputer is used, would have to be developed so the proposed hardware could successfully communicate with the minicomputer.

Using the special hardware instead of a program to form a decomposition chart would be about 2^n times faster, where n is the number of variables of the function being investigated. This estimate is based on the fact that a decomposition chart can be formed in one clock cycle with the special hardware, while each minterm would have to be moved individually in a program. Assuming one move can be made each clock cycle, 2^n colck cycles would be required. Finding the column multiplicity of a decomposition chart using the special hardware instead of a program would be about as many times faster as there are columns on the chart under consideration. This is because the special hardware compares a column with all the other columns in parallel, requiring only as many clock cycles to complete the examination as there are different columns. A program would have to compare each column

individually with the allowed choices to find out with which one it agreed. If no agreement was found, a new choice would have to be created. In the worst case, the comparisons would require a number of clock cycles given approximately by (number of different columns) x (total number of columns), assuming one comparison can be made in each clock cycle.

The cost of the hardware required to investigate a function of n variables is roughly summarized in Table 3. For 6 variables, 114 flip flops would be needed, which is roughly equivalent to seven 16 bit registers. 32 EQUIVALENCE FUNCTION gates would be needed. Since an EQUIVALENCE FUNCTION gate is about the same complexity as an EXCLUSIVE OR, which is a half adder, the cost of the Equivalence Functions is about the same as the cost of two 16 bit full adders. 128 AND-OR combinations having 4 AND gates each would be needed. Assuming 4 new shifts are needed, approximately 425 AND gates would be needed. All totaled, this involves quite a bit of hardware, but probably is only a small fraction of the hardware in a typical minicomputer.

Hardware	Purpose
Flip Flops	
2^n	Minterm Register
3n	Variable Order Rearranger
$2^n/2$	Column Multiplicity Recognizer
AND Gates	
2^n+3n	Minterm Rearranger, for each shift
$3(2^n/2)$	Column Multiplicity Recognizer
AND-OR Combinations	
2^n	Output Bus Gates, one AND required for each column length
2^n	Input Bus Gates, one AND required for each column length
Equivalence Functions	
2^n	Column Multiplicity Recognizer

Plus a few small counters, instruction registers, and decoders

Table 3. Rough cost estimate of proposed hardware to investigate a function of n variables.

LIST OF REFERENCES

1. R. L. Ashenhurst, "The Decomposition of Switching Functions," Proc. Internatl. Symp. Theory of Switching (April 2-5, 1957), Vol. 29 of Annals of Computation Laboratory of Harvard University, Cambridge, Mass., 1959, pp. 74-116. (Reprinted in Appendix of (5).)

2. R. M. Karp, F. E. McFarlin, J. P. Roth, and J. R. Wilts, "A Computer Program for the Synthesis of Combinational Switching Circuits," Proc. Second Annual AIEE Symposium on Switching Circuit Theory and Logical Design, October 1961, pp. 182-194.

3. P. R. Schneider and D. L. Dietmeyer, "An Algorithm for Synthesis of Multiple-Output Combinational Logic," IEEE Transactions on Computers, Vol. C-17, No. 2, pp. 117-128, February 1968.

4. V. Yun-Shen Shen and A. C. McKellar, "An Algorithm for the Disjunctive Decomposition of Switching Functions," IEEE Transactions on Computers, Vol. C-19, No. 3, March 1970, pp. 239-248.

5. H. A. Curtis, A New Approach to the Design of Switching Circuits, D. Van Nostrand Co., Inc., 1962.

6. R. M. Karp, "Functional Decomposition and Switching Circuit Design," J. Soc. Indust. Appl. Math., Vol. II, No. 2, pp. 291-335, June 1963.

7. M. A. Marin, "On a General Synthesis Algorithm of Logical Circuits using a Restricted Inventory of Integrated Circuits," Proc. of the Share ACM-IEEE Design Automation Workshop, July 15-18, 1968, pp. 4-1 to 4-38.

8. E. S. Davidson, "An Algorithm for NAND Decomposition under Network Constraints," IEEE Transactions on Computers, Vol. C-18, No. 12, pp. 1098-1109, December 1969.

9. H. A. Curtis, "A Generalized Tree Circuit," ACM Journal, Vol. 8, No. 4, pp. 484-496, October 1961.

10. H. A. Curtis, "Generalized Tree Circuit--The Basic Building Block of an Extended Decomposition Theory," ACM Journal, Vol. 10, No. 4, October 1963.

11. S. M. Johnson, "Generation of Permutations by Adjacent Transportation," Math Comp., Vol. 17, 1963 , 282-285.

APPENDIX A

COMPUTER PROGRAMS AND DISCUSSION OF RESULTS FOR MINTERM REARRANGER SHIFTS AND CYCLES OF SHIFTS

The computer programs shown here generate trees to investigate cycles of shifts for the minterm rearranger, as discussed in Section 3.2. Investigations have been limited to four variables, but the method is general and may be extended to more than four variables. From Table 1, the charts needed for four variables are -/3/1(4), -/2/2(6), and 1/2/1(12). The shifts available for consideration are the 18 true four-variable shifts at a cost of 2, and the extended three-variable shift at a cost of 1. The cheapest cycles to generate the desired variable orders for each of the different types of charts will be sought.

The program of Figure 16 investigates the -/3/1(4) charts using a shift library of all 19 shifts. Six realizations were found at a cost of 2 and four realizations were found at a cost of 3. The program of Figure 17 investigates the -/2/2(6) charts. Those shifts not useful in forming the -/3/1(4) charts were eliminated from consideration. This left only shifts 1, 3, 6, 7, 8, 10, 12, 15, and 16. No realizations were found at a cost of 2 and no realizations were found at a cost of 3. If a different set of shifts had been used, no doubt some realizations at a cost of 3 could be obtained. However, the set of shifts used here are the only shifts which can be used to form the -/3/1(4) charts at a cost less than or equal to 3. This proves that the cost of realizing both the -/3/1(4) charts and the -/2/2(6) using the same shifts must be greater than 3.

To find all possible solutions of cost 4, it would be necessary to go back and find the set of shifts which can realize the -/3/1(4) charts at a cost of 4. Then these could be used to investigate the -/2/2(6) and the 1/2/1(12) charts. Only some of the solutions will be investigated here. The backtrack to the cost 4,-/3/1(4) charts will not be done. Instead, only those realizations of cost 4 shown in Figure 17 using the cost 3,-/3/1(4) shift library will be considered. This limited shift library is ample to give many usable realizations for the 1/2/1(12) charts and the number of solutions does not become so large that they cannot be included here.

One backtrack is necessary, however. This comes about because cost 4 realizations correspond to implementing two new four-variable shifts while cost 3 realizations correspond to implementing one new four-variable shift and extending the three-variable shift. The extended shift used in cost 3 realizations is not available with cost 4 realizations. Hence, cost 3 realizations are not usable when the upper cost bound is 4. Cost 2 realizations are still allowed because they correspond to implementing a single new four variable shift. From Figure 16, it can be seen that shifts 1, 6, and 10 result in cost 3 realizations for the -/3/1(4) charts and do not appear in the cost 2 realizations. Since the backtrack to determine whether they would appear in the cost 4 realizations is not going to be done, to be safe these shifts will be removed from the library before investigating the 1/2/1(12) charts. Cost 4 realizations numbered 5 and 18 in Figure 17 for the -/2/2(6) charts are actually invalid because they are based on shifts

6 and 10. There is no chance they would be used in a final solution, however, since the illegal shifts will be removed before the 1/2/1(12) charts will be investigated.

The program of Figure 18 investigates the 1/2/1(12) charts. The shifts considered are those from the cost 3,-/3/1(4) charts, excluding shifts 1, 6, and 10. All of these shifts appear in the cost 4,-/2/2(6) realizations. Of the 32 realizations obtained, 16 use a combination of two shifts which would also be usable in realizing the -/2/2(6) charts. These combinations are 3, 12; 3, 15; 8, 12; and 8, 15. For any one of these pairs of shifts, a solution for the four-variable charts is possible. Assume, for example, that shifts 3 and 12 were to be implemented. Then the 1/2/1(12) charts could be generated by any of the cost 4 realizations numbered 5, 8, 13, and 32 in Figure 18. The -/2/2(6) charts could be generated by either of the cost 4 realizations numbered 2 and 11 in Figure 17. The -/3/1(4) charts could be generated by either of the cost 2 realizations numbered 3 and 5 in Figure 16.

The minterm orders implied by the variable orders of the shifts in the cost 3,-/3/1(4) library are shown in Figure 19 of Appendix B. The implied minterm order is all that is needed to implement a shift, as discussed in Section 3.2.

```
TREE: PROC OPTIONS (MAIN);

        /* NOTE: IN THIS PROGRAM A SHIFT HAS BEEN CALLED A GATE */
              /* OTHER MNEMONICS ARE FAIRLY OBVIOUS */
        /* OUTPUT WILL BE PUNCHED */

DCL (VAR(5), VARORD(5), I(5), II(5), COST1, COMP, CK, D,
   SEP, NAV, NO, GA, TGT, TPCOV, TCORD, TUORD(5),COST) FIXED BIN;
DCL GATE(19) LABEL INITIAL (CG1,CG2,CG3,CG4,CG5,CG6,CG7,CG8,CG9,
   CG10,CG11,CG12,CG13,CG14,CG15,CG16,CG17,CG18,CG19);
DCL CP ENTRY ((*) FIXED BIN,FIXED BIN) RETURNS (FIXED BIN);
DCL GG ENTRY ((*) FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,
   FIXED BIN);
DCL (PART(12,6 ), PCOV(1000,0:12), GT(1000,0:12),
     CORD(1000,0:12), UORD(1000,5)) FIXED BIN;

        /* COMPACT TO ONE WORD */
CP: PROC (VARORD,N) RETURNS(FIXED BIN);
DCL (VARORD(5),COMP) FIXED BIN;
   COMP=VARORD(1);
DO KK=2 TO N;
COMP=(10**(KK-1))*VARORD(KK)+COMP;
   END;
   RETURN (COMP);
   END CP;

        /* SIMULATION GATE */
GG: PROC (VARORD,IE,ID,IC,IB,IA);
DCL (VARORD(5), VAR(5)) FIXED BINARY;
VAR=VARORD;
VARORD(1)=VAR(IA);
VARORD(2)=VAR(IB);
VARORD(3)=VAR(IC);
VARORD(4)=VAR(ID);
VARORD(5)=VAR(IE);
END GG;

        /* CONTROL CARD  */
N=4; NAV=1000; GA=19; D=4; SEP=6; COST1=1; MAXCOST=3; /* -/3/1 */

        /* GENERATE PARTITIONS */
        /* INITIALIZE */
VARORD=0;
DO I(1)=1 TO N;
VAR(I(1))=I(1);
END;
PUT FILE(SYSPNCH) EDIT ('-/3/1 (4)') (SKIP,A(9));
K=0;      /* K COUNTS NUMBER OF PARTITIONS */
```

Figure 16. Program for -/3/1(4) Charts

```
          /* PICK ORDER FOR KTH PARTITION */
L031A: DO I(1)=1 TO N;
L031B: DO I(2)=1 TO N;
IF I(2)=I(1) THEN GO TO EL031B;
L031C: DO I(3)=I(2)+1 TO N;
IF (I(3)=I(1))|(I(3)=I(2)) THEN GO TO EL031C;
L031D: DO I(4)=I(3)+1 TO N;
IF (I(4)=I(1))|(I(4)=I(2))|(I(4)=I(3)) THEN GO TO EL031D;
K=K+1;
M=0;      /* M COUNTS SIZE OF EACH PARTITION */
          /* FORM ALL PERMUTATIONS OF ORIGINAL ORDER
            TO CREATE PARTITION */
PUT FILE(SYSPNCH) EDIT('PARTITION NUMBER',K) (SKIP,A(16),F(3));
VARORD(1)=VAR(I(1));
L031BB: DO II(2)=2,3,4;
L031CC: DO II(3)=2,3,4;
IF II(3)=II(2) THEN GO TO EL031CC;
L031DD: DO II(4)=2,3,4;
IF (II(4)=II(2))|(II(4)=II(3)) THEN GO TO EL031DD;
VARORD(2)=VAR(I(II(2)));
VARORD(3)=VAR(I(II(3)));
VARORD(4)=VAR(I(II(4)));
M=M+1;
          /* COMPACT, STORE, AND PRINT */
PART(K,M)=CP(VARORD,N);
PUT FILE(SYSPNCH) EDIT (PART(K,M)) (SKIP,F(5));
EL031DD: END L031DD;
EL031CC: END L031CC;
EL031BB: END L031BB;
EL031D : END L031D ;
EL031C : END L031C ;
EL031B : END L031B ;
EL031A : END L031A ;

          /* GENERATE TREE TO INVESTIGATE GATES */
          /* INITIALIZE */
GT=-1; PCOV=0; TUORD=0; UORD=0; VAR=0;
DO K=1 TO N;
UORD(1,K)=K; TUORD(K)=K;
END;
L=0; NO=1;
CORD(NO,L)=CP(TUORD,N);
          /* 0TH ELEMENT OF GT WILL BE USED AS A FLAG TO TELL WHAT
             LEVEL THE ARRAY IS ON */
GT(NO,0)=0;
```

Figure 16. Program for -/3/1(4) Charts

```
        /* MOVE TO NEXT LEVEL ON TREE. L CONTAINS CURRENT LEVEL */
  NLEV: M=0;
DO K=1 TO NAV;
IF GT(K,0)=L THEN M=M+1;
END;
PUT FILE(SYSPNCH) EDIT('LEVEL',L) (SKIP,A(5),F(3));
PUT FILE(SYSPNCH) EDIT ('NUMBER OF NODES ON TREE ARE',M)
(SKIP,A(27),F(5));
IF M>0 THEN GO TO INCRL;
PUT FILE(SYSPNCH) EDIT ('NO REALIZATION IS POSSIBLE')
(SKIP,A(26));
GO TO ETR;
  INCRL: L=L+1;
IF L>D THEN DO; L=D; GO TO PRNT; END;
NO=0;
        /* FIND NEXT AVAILABLE ARRAY TO BE INVESTIGATED. NO CONTAINS
           THE NUMBER OF THE ARRAY CURRENTLY UNDER INVESTIGATION */
  NXAV: NO=NO+1;
IF NO>NAV THEN GO TO NLEV;
IF GT(NO,0)¬=L-1 THEN GO TO NXAV;
DO K=1 TO N;
VAR(K)=UORD(NO,K);
END;
TGT=0;
        /* SELECT A TEMPORARY GATE TO BE CALLED.  TGT CONTAINS THE
           SUBSCRIPT OF THE GATE TO BE CALLED. COST CONTAINS THE
           COST OF CALLING THE GATE.  THE TOTAL COST SO FAR IS
           STORED AS A FLAG IN THE 0TH ELEMENT OF PCOV */
  CGT: TGT=TGT+1;
IF TGT>GA THEN DO; GT(NO,0)=-1; GO TO NXAV; END;
TUORD=VAR;
COST=0;
DO K=1 TO L-1;
IF GT(NO,K)=TGT THEN GO TO GATE(TGT);
END;
IF TGT<=COST1 THEN COST=1; ELSE COST=2;
IF PCOV(NO,0)+COST>MAXCOST THEN GO TO CGT;
GO TO GATE(TGT);
        /* ALL GATES */
CG1 : CALL GG( TUORD,5,4,2,1,3);   GO TO FPCOV;
CG2 : CALL GG( TUORD,5,1,2,3,4);   GO TO FPCOV;
CG3 : CALL GG( TUORD,5,1,2,4,3);   GO TO FPCOV;
CG4 : CALL GG( TUORD,5,1,3,2,4);   GO TO FPCOV;
CG5 : CALL GG( TUORD,5,1,3,4,2);   GO TO FPCOV;
CG6 : CALL GG( TUORD,5,1,4,2,3);   GO TO FPCOV;
CG7 : CALL GG( TUORD,5,1,4,3,2);   GO TO FPCOV;
CG8 : CALL GG( TUORD,5,2,1,3,4);   GO TO FPCOV;
CG9 : CALL GG( TUORD,5,2,1,4,3);   GO TO FPCOV;
```

Figure 16. Program for -/3/1(4) Charts

```
CG10: CALL GG( TUORD,5,2,3,1,4);    GO TO FPCOV;
CG11: CALL GG( TUORD,5,2,3,4,1);    GO TO FPCOV;
CG12: CALL GG( TUORD,5,2,4,1,3);    GO TO FPCOV;
CG13: CALL GG( TUORD,5,2,4,3,1);    GO TO FPCOV;
CG14: CALL GG( TUORD,5,3,1,2,4);    GO TO FPCOV;
CG15: CALL GG( TUORD,5,3,1,4,2);    GO TO FPCOV;
CG16: CALL GG( TUORD,5,3,2,1,4);    GO TO FPCOV;
CG17: CALL GG( TUORD,5,3,2,4,1);    GO TO FPCOV;
CG18: CALL GG( TUORD,5,3,4,1,2);    GO TO FPCOV;
CG19: CALL GG( TUORD,5,3,4,2,1);    GO TO FPCOV;
          /* FIND PARTITION COVERED BY NEW VARIABLE ORDER */
  FPCOV: TCORD=CP(TUORD,N);
LOOP: DO K=1 TO D;
DO M=1 TO SEP;
IF TCORD=PART(K,M) THEN DO;
TPCOV=K;
GO TO CKPCOV; END;
END LOOP;
          /* CHECK PARTITION COVERED FOR DUPLICATION */
  CKPCOV: DO K=1 TO L-1;
IF TPCOV=PCOV(NO,K) THEN GO TO CGT;
END CKPCOV;
          /* FIND A VACANT ARRAY.  K CONTAINS THE SUBSCRIPT OF THE
             VACANT ARRAY FOUND */
  FVAC: DO K=1 TO NAV;
IF GT(K,0)=-1 THEN GO TO ADD;
END FVAC;
PUT FILE(SYSPNCH) EDIT ('NEED MORE ROOM LEVEL',L)
(SKIP,A(20),F(3));
PUT FILE(SYSPNCH) EDIT ('PROCESSING NUMBER',NO)
(SKIP,A(17),F(3));
GO TO ETR;
          /* TRANSFER OLD ARRAY INTO VACANT ARRAY AND ADD NEXT LEVEL*/
  ADD: DO M=0 TO L-1;
GT(K,M)=GT(NO,M);
PCOV(K,M)=PCOV(NO,M);
CORD(K,M)=CORD(NO,M);
END;
GT(K,0)=L;
PCOV(K,0)=PCOV(NO,0)+COST;
DO M=1 TO N;
UORD(K,M)=TUORD(M);
END;
GT(K,L)=TGT;
PCOV(K,L)=TPCOV;
CORD(K,L)=TCORD;
GO TO CGT;
```

Figure 16. Program for -/3/1(4) Charts

```
        /* PRINT OUT ALL FINAL NODES WHICH RESTORE TO
           ORIGINAL ORDER */
  PRNT: DO COST=2 TO MAXCOST;
NUM1=-1; NUM2=0; NN=1;
PUT FILE(SYSPNCH) EDIT ('COST=',COST) (SKIP,A(5),F(3));
POUT: DO K=1 TO NAV;
IF GT(K,0)<L   THEN GO TO EPOUT;
IF PCOV(K,0)¬=COST THEN GO TO EPOUT;
IF CORD(K,D)¬=CORD(K,0) THEN GO TO EPOUT;
IF NN=1 THEN DO;
NUM1=NUM1+2;
NN=2;  K1=K;
GO TO EPOUT;  END;
NUM2=NUM2+2;
NN=1;  K2=K;
PUT FILE(SYSPNCH) EDIT ('REALIZATION NUMBER',NUM1,
'REALIZATION NUMBER',NUM2) (SKIP,A(18),F(3),X(14),A(18),F(3));
PUT FILE(SYSPNCH) EDIT ('SHIFT','ORDER','PARTITION','SHIFT',
'ORDER','PARTITION') (SKIP,A( 5),X(2),A(5),X(2),A(9),X(12),
A(5),X( 2),A(5),X(2),A(9));
PUT FILE(SYSPNCH) EDIT(CORD(K1,0),CORD(K2,0))
(SKIP,X(4),F(8),X(27),F(8));
DO M=1 TO D;
PUT FILE(SYSPNCH) EDIT (GT(K1,M),CORD(K1,M),PCOV(K1,M),
GT(K2,M),CORD(K2,M),PCOV(K2,M))
(SKIP,F(4),F(8),F(7),X(16),F(4),F(8),F(7));
END;
EPOUT: END POUT;
IF NUM1=-1 THEN DO;
PUT FILE(SYSPNCH) EDIT ('NO REALIZATIONS') (SKIP,A(15)); END;
IF NN=2 THEN DO;
PUT FILE(SYSPNCH) EDIT ('REALIZATION NUMBER',NUM1)
(SKIP,A(18),F(3));
PUT FILE(SYSPNCH) EDIT ('SHIFT','ORDER','PARTITION')
(SKIP,A(5),X(2),A(5),X(2),A(9));
PUT FILE(SYSPNCH) EDIT(CORD(K1,0)) (SKIP,X(4),F(8));
DO M=1 TO D;
PUT FILE(SYSPNCH) EDIT (GT(K1,M),CORD(K1,M),PCOV(K1,M))
(SKIP,F(4),F(8),F(7));
END; END;
END PRNT;
ETR: END TREE;
```

Figure 16. Program for -/3/1(4) Charts

```
-/3/1 (4)

PARTITION NUMBER  1
 4321
 3421
 4231
 2431
 3241
 2341

PARTITION NUMBER  2
 4312
 3412
 4132
 1432
 3142
 1342

PARTITION NUMBER  3
 4213
 2413
 4123
 1423
 2143
 1243

PARTITION NUMBER  4
 3214
 2314
 3124
 1324
 2134
 1234
```

Figure 16. Program for -/3/1(4) Charts

```
LEVEL  0
NUMBER OF NODES ON TREE ARE     1

LEVEL  1
NUMBER OF NODES ON TREE ARE    19

LEVEL  2
NUMBER OF NODES ON TREE ARE    47

LEVEL  3
NUMBER OF NODES ON TREE ARE    79

LEVEL  4
NUMBER OF NODES ON TREE ARE    68
```

Figure 16. Program for -/3/1(4) Charts

COST= 2

REALIZATION NUMBER 1

SHIFT	ORDER	PARTITION
	4321	
8	2134	4
8	3412	2
8	1243	3
8	4321	1

REALIZATION NUMBER 2

SHIFT	ORDER	PARTITION
	4321	
15	3142	2
15	1234	4
15	2413	3
15	4321	1

REALIZATION NUMBER 3

SHIFT	ORDER	PARTITION
	4321	
3	1243	3
3	3412	2
3	2134	4
3	4321	1

REALIZATION NUMBER 4

SHIFT	ORDER	PARTITION
	4321	
7	1432	2
7	2143	3
7	3214	4
7	4321	1

REALIZATION NUMBER 5

SHIFT	ORDER	PARTITION
	4321	
12	2413	3
12	1234	4
12	3142	2
12	4321	1

REALIZATION NUMBER 6

SHIFT	ORDER	PARTITION
	4321	
16	3214	4
16	2143	3
16	1432	2
16	4321	1

Figure 16. Program for -/3/1(4) Charts

COST= 3

REALIZATION NUMBER 1

SHIFT	ORDER	PARTITION
	4321	
1	4213	3
6	3412	2
1	3124	4
6	4321	1

REALIZATION NUMBER 2

SHIFT	ORDER	PARTITION
	4321	
1	4213	3
10	1234	4
1	1342	2
10	4321	1

REALIZATION NUMBER 3

SHIFT	ORDER	PARTITION
	4321	
6	1423	3
1	1234	4
6	4132	2
1	4321	1

REALIZATION NUMBER 4

SHIFT	ORDER	PARTITION
	4321	
10	2314	4
1	2143	3
10	4132	2
1	4321	1

Figure 16. Program for -/3/1(4) Charts

```
TREE: PROC OPTIONS (MAIN);

        /* NOTE: IN THIS PROGRAM A SHIFT HAS BEEN CALLED A GATE */
              /* OTHER MNEMONICS ARE FAIRLY OBVIOUS */
        /* OUTPUT WILL BE PUNCHED */

DCL (VAR(5), VARORD(5), I(5), II(5), COST1, COMP, CK, D,
   SEP, NAV, NO, GA, TGT, TPCOV, TCORD, TUORD(5),COST) FIXED BIN;
DCL GATE(19) LABEL INITIAL (CG1,CG2,CG3,CG4,CG5,CG6,CG7,CG8,CG9,
   CG10,CG11,CG12,CG13,CG14,CG15,CG16,CG17,CG18,CG19);
DCL CP ENTRY ((*) FIXED BIN,FIXED BIN) RETURNS (FIXED BIN);
DCL GG ENTRY ((*) FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,
   FIXED BIN);
DCL (PART(12,6 ), PCOV(1000,0:12), GT(1000,0:12),
     CORD(1000,0:12), UORD(1000,5)) FIXED BIN;

        /* COMPACT TO ONE WORD */
CP: PROC (VARORD,N) RETURNS(FIXED BIN);
DCL (VARORD(5),COMP) FIXED BIN;
   COMP=VARORD(1);
DO KK=2 TO N;
COMP=(10**(KK-1))*VARORD(KK)+COMP;
   END;
   RETURN (COMP);
   END CP;

        /* SIMULATION GATE */
GG: PROC (VARORD,IE,ID,IC,IB,IA);
DCL (VARORD(5), VAR(5)) FIXED BINARY;
VAR=VARORD;
VARORD(1)=VAR(IA);
VARORD(2)=VAR(IB);
VARORD(3)=VAR(IC);
VARORD(4)=VAR(ID);
VARORD(5)=VAR(IE);
END GG;

        /* CONTROL CARD  */
N=4; NAV=1000; GA=19; D=6; SEP=4; COST1=1; MAXCOST=4; /* -/2/2 */

        /* GENERATE PARTITIONS */
        /* INITIALIZE */
VARORD=0;
DO I(1)=1 TO N;
VAR(I(1))=I(1);
END;
PUT FILE(SYSPNCH) EDIT ('-/2/2 (6)') (SKIP,A(9));
K=0;     /* K COUNTS NUMBER OF PARTITIONS */
```

Figure 17. Program for -/2/2(6) Charts

```
          /* PICK ORDER FOR KTH PARTITION */
L022A: DO I(1)=1 TO N;
L022B: DO I(2)= (I(1)+1) TO N;
L022C: DO I(3)=1 TO N;
IF (I(3)=I(1))|(I(3)=I(2)) THEN GO TO EL022C;
L022D: DO I(4)=I(3)+1 TO N;
IF (I(4)=I(1))|(I(4)=I(2)) THEN GO TO EL022D;
K=K+1;
M=0;      /* M COUNTS SIZE OF EACH PARTITION */
          /* FORM ALL PERMUTATIONS OF ORIGINAL ORDER
            TO CREATE PARTITION */
PUT FILE(SYSPNCH) EDIT('PARTITION NUMBER',K) (SKIP,A(16),F(3));
L022AA: DO II(1)=1,2;
L022BB: DO II(2)=1,2;
IF II(2)=II(1) THEN GO TO EL022BB;
L022CC: DO II(3)=3,4;
L022DD: DO II(4)=3,4;
IF II(3)=II(4) THEN GO TO EL022DD;
VARORD(1)=VAR(I(II(1)));
VARORD(2)=VAR(I(II(2)));
VARORD(3)=VAR(I(II(3)));
VARORD(4)=VAR(I(II(4)));
          /* COMPACT, STORE, AND PRINT */
M=M+1;
PART(K,M)=CP(VARORD,N);
PUT FILE(SYSPNCH) EDIT (PART(K,M)) (SKIP,F(5));
EL022DD: END L022DD;
EL022CC: END L022CC;
EL022BB: END L022BB;
EL022AA: END L022AA;
EL022D : END L022D ;
EL022C : END L022C ;
EL022B : END L022B ;
EL022A : END L022A ;

          /* GENERATE TREE TO INVESTIGATE GATES */
          /* INITIALIZE */
GT=-1; PCOV=0; TUORD=0; UORD=0; VAR=0;
DO K=1 TO N;
UORD(1,K)=K; TUORD(K)=K;
END;
L=0; NO=1;
CORD(NO,L)=CP(TUORD,N);
          /* 0TH ELEMENT OF GT WILL BE USED AS A FLAG TO TELL WHAT
             LEVEL THE ARRAY IS ON */
GT(NO,0)=0;
```

Figure 17. Program for -/2/2(6) Charts

```
        /* MOVE TO NEXT LEVEL ON TREE. L CONTAINS CURRENT LEVEL */
  NLEV: M=0;
DO K=1 TO NAV;
IF GT(K,0)=L THEN M=M+1;
END;
PUT FILE(SYSPNCH) EDIT('LEVEL',L) (SKIP,A(5),F(3));
PUT FILE(SYSPNCH) EDIT ('NUMBER OF NODES ON TREE ARE',M)
(SKIP,A(27),F(5));
IF M>0 THEN GO TO INCRL;
PUT FILE(SYSPNCH) EDIT ('NO REALIZATION IS POSSIBLE')
(SKIP,A(26));
GO TO ETR;
  INCRL: L=L+1;
IF L>D THEN DO; L=D; GO TO PRNT; END;
NO=0;
        /* FIND NEXT AVAILABLE ARRAY TO BE INVESTIGATED. NO CONTAINS
           THE NUMBER OF THE ARRAY CURRENTLY UNDER INVESTIGATION */
  NXAV: NO=NO+1;
IF NO>NAV THEN GO TO NLEV;
IF GT(NO,0)¬=L-1 THEN GO TO NXAV;
DO K=1 TO N;
VAR(K)=UORD(NO,K);
END;
TGT=0;
        /* SELECT A TEMPORARY GATE TO BE CALLED.  TGT CONTAINS THE
           SUBSCRIPT OF THE GATE TO BE CALLED. COST CONTAINS THE
           COST OF CALLING THE GATE.  THE TOTAL COST SO FAR IS
           STORED AS A FLAG IN THE 0TH ELEMENT OF PCOV */
  CGT: TGT=TGT+1;
IF TGT>GA THEN DO; GT(NO,0)=-1; GO TO NXAV; END;
TUORD=VAR;
COST=0;
DO K=1 TO L-1;
IF GT(NO,K)=TGT THEN GO TO GATE(TGT);
END;
IF TGT<=COST1 THEN COST=1; ELSE COST=2;
IF PCOV(NO,0)+COST>MAXCOST THEN GO TO CGT;
GO TO GATE(TGT);
        /* ONLY GATES 1 3 6 7 8 10 12 15 16 */
CG1 : CALL GG( TUORD,5,4,2,1,3);   GO TO FPCOV;
CG2 : GO TO CGT;
CG3 : CALL GG( TUORD,5,1,2,4,3);   GO TO FPCOV;
CG4 : GO TO CGT;
CG5 : GO TO CGT;
CG6 : CALL GG( TUORD,5,1,4,2,3);   GO TO FPCOV;
CG7 : CALL GG( TUORD,5,1,4,3,2);   GO TO FPCOV;
CG8 : CALL GG( TUORD,5,2,1,3,4);   GO TO FPCOV;
CG9 : GO TO CGT;
```

Figure 17. Program for -/2/2(6) Charts

```
CG10: CALL GG( TUORD,5,2,3,1,4);   GO TO FPCOV;
CG11: GO TO CGT;
CG12: CALL GG( TUORD,5,2,4,1,3);   GO TO FPCOV;
CG13: GO TO CGT;
CG14: GO TO CGT;
CG15: CALL GG( TUORD,5,3,1,4,2);   GO TO FPCOV;
CG16: CALL GG( TUORD,5,3,2,1,4);   GO TO FPCOV;
CG17: GO TO CGT;
CG18: GO TO CGT;
CG19: GO TO CGT;
         /* FIND PARTITION COVERED BY NEW VARIABLE ORDER */
  FPCOV: TCORD=CP(TUORD,N);
LOOP: DO K=1 TO D;
DO M=1 TO SEP;
IF TCORD=PART(K,M) THEN DO;
TPCOV=K;
GO TO CKPCOV; END;
END LOOP;
         /* CHECK PARTITION COVERED FOR DUPLICATION */
  CKPCOV: DO K=1 TO L-1;
IF TPCOV=PCOV(NO,K) THEN GO TO CGT;
END CKPCOV;
         /* FIND A VACANT ARRAY.  K CONTAINS THE SUBSCRIPT OF THE
            VACANT ARRAY FOUND */
  FVAC: DO K=1 TO NAV;
IF GT(K,0)=-1 THEN GO TO ADD;
END FVAC;
PUT FILE(SYSPNCH) EDIT ('NEED MORE ROOM LEVEL',L)
(SKIP,A(20),F(3));
PUT FILE(SYSPNCH) EDIT ('PROCESSING NUMBER',NO)
(SKIP,A(17),F(3));
GO TO ETR;
         /* TRANSFER OLD ARRAY INTO VACANT ARRAY AND ADD NEXT LEVEL*/
  ADD: DO M=0 TO L-1;
GT(K,M)=GT(NO,M);
PCOV(K,M)=PCOV(NO,M);
CORD(K,M)=CORD(NO,M);
END;
GT(K,0)=L;
PCOV(K,0)=PCOV(NO,0)+COST;
DO M=1 TO N;
ORD(K,M)=TUORD(M);
END;
T(K,L)=TGT;
COV(K,L)=TPCOV;
ORD(K,L)=TCORD;
O TO CGT;
```

Figure 17. Program for -/2/2(6) Charts

```
          /* PRINT OUT ALL FINAL NODES WHICH RESTORE TO
             ORIGINAL ORDER */
   PRNT: DO COST=2 TO MAXCOST;
NUM1=-1; NUM2=0; NN=1;
PUT FILE(SYSPNCH) EDIT ('COST=',COST) (SKIP,A(5),F(3));
POUT: DO K=1 TO NAV;
IF GT(K,0)<L    THEN GO TO EPOUT;
IF PCOV(K,0)¬=COST THEN GO TO EPOUT;
IF CORD(K,D)¬=CORD(K,0) THEN GO TO EPOUT;
IF NN=1 THEN DO;
NUM1=NUM1+2;
NN=2;  K1=K;
GO TO EPOUT;  END;
NUM2=NUM2+2;
NN=1;  K2=K;
PUT FILE(SYSPNCH) EDIT ('REALIZATION NUMBER',NUM1,
'REALIZATION NUMBER',NUM2) (SKIP,A(18),F(3),X(14),A(18),F(3));
PUT FILE(SYSPNCH) EDIT ('SHIFT','ORDER','PARTITION','SHIFT',
'ORDER','PARTITION') (SKIP,A( 5),X(2),A(5),X(2),A(9),X(12),
A(5),X( 2),A(5),X(2),A(9));
PUT FILE(SYSPNCH) EDIT(CORD(K1,0),CORD(K2,0))
(SKIP,X(4),F(8),X(27),F(8));
DO M=1 TO D;
PUT FILE(SYSPNCH) EDIT (GT(K1,M),CORD(K1,M),PCOV(K1,M),
GT(K2,M),CORD(K2,M),PCOV(K2,M))
(SKIP,F(4),F(8),F(7),X(16),F(4),F(8),F(7));
END;
EPOUT: END POUT;
IF NUM1=-1 THEN DO;
PUT FILE(SYSPNCH) EDIT ('NO REALIZATIONS') (SKIP,A(15)); END;
IF NN=2 THEN DO;
PUT FILE(SYSPNCH) EDIT ('REALIZATION NUMBER',NUM1)
(SKIP,A(18),F(3));
PUT FILE(SYSPNCH) EDIT ('SHIFT','ORDER','PARTITION')
(SKIP,A(5),X(2),A(5),X(2),A(9));
PUT FILE(SYSPNCH) EDIT(CORD(K1,0)) (SKIP,X(4),F(8));
DO M=1 TO D;
PUT FILE(SYSPNCH) EDIT (GT(K1,M),CORD(K1,M),PCOV(K1,M))
(SKIP,F(4),F(8),F(7));
END; END;
END PRNT;
ETR: END TREE;
```

Figure 17. Program for -/2/2(6) Charts

```
-/2/2 (6)


PARTITION NUMBER  1
 4321
 3421
 4312
 3412

PARTITION NUMBER  2
 4231
 2431
 4213
 2413

PARTITION NUMBER  3
 3241
 2341
 3214
 2314

PARTITION NUMBER  4
 4132
 1432
 4123
 1423

PARTITION NUMBER  5
 3142
 1342
 3124
 1324

PARTITION NUMBER  6
 2143
 1243
 2134
 1234
```

Figure 17. Program for -/2/2(6) Charts

```
LEVEL  0
NUMBER OF NODES ON TREE ARE     1

LEVEL  1
NUMBER OF NODES ON TREE ARE     9

LEVEL  2
NUMBER OF NODES ON TREE ARE    81

LEVEL  3
NUMBER OF NODES ON TREE ARE   181

LEVEL  4
NUMBER OF NODES ON TREE ARE   250

LEVEL  5
NUMBER OF NODES ON TREE ARE   220

LEVEL  6
NUMBER OF NODES ON TREE ARE    52
```

Figure 17. Program for -/2/2(6) Charts

COST= 2
NO REALIZATIONS

COST= 3
NO REALIZATIONS

COST= 4

REALIZATION NUMBER 1

SHIFT	ORDER	PARTITION
	4321	
3	1243	6
7	3124	5
3	4231	2
7	1423	4
3	3214	3
7	4321	1

REALIZATION NUMBER 2

SHIFT	ORDER	PARTITION
	4321	
3	1243	6
12	4132	4
3	2341	3
12	4213	2
3	3142	5
12	4321	1

REALIZATION NUMBER 3

SHIFT	ORDER	PARTITION
	4321	
3	1243	6
15	2314	3
3	4123	4
15	1342	5
3	2413	2
15	4321	1

REALIZATION NUMBER 4

SHIFT	ORDER	PARTITION
	4321	
3	1243	6
16	2431	2
3	1324	5
16	3241	3
3	1432	4
16	4321	1

Figure 17. Program for -/2/2(6) Charts

REALIZATION NUMBER 5

SHIFT	ORDER	PARTITION
	4321	
6	1423	4
10	2431	2
6	1234	6
10	3241	3
6	1342	5
10	4321	1

REALIZATION NUMBER 6

SHIFT	ORDER	PARTITION
	4321	
7	1432	4
3	2314	3
7	4231	2
3	1342	5
7	2134	6
3	4321	1

REALIZATION NUMBER 7

SHIFT	ORDER	PARTITION
	4321	
7	1432	4
8	3241	3
7	1324	5
8	2431	2
7	1243	6
8	4321	1

REALIZATION NUMBER 8

SHIFT	ORDER	PARTITION
	4321	
8	2134	6
12	3241	3
8	4123	4
12	2431	2
8	3142	5
12	4321	1

REALIZATION NUMBER 9

SHIFT	ORDER	PARTITION
	4321	
8	2134	6
15	1423	4
8	2341	3
15	3124	5
8	2413	2
15	4321	1

REALIZATION NUMBER 10

SHIFT	ORDER	PARTITION
	4321	
8	2134	6
16	1342	5
8	4231	2
16	2314	3
8	1432	4
16	4321	1

REALIZATION NUMBER 11

SHIFT	ORDER	PARTITION
	4321	
12	2413	2
3	3124	5
12	2341	3
3	1423	4
12	2134	6
3	4321	1

REALIZATION NUMBER 12

SHIFT	ORDER	PARTITION
	4321	
12	2413	2
8	1342	5
12	4123	4
8	2314	3
12	1243	6
8	4321	1

Figure 17. Program for -/2/2(6) Charts

REALIZATION NUMBER 13

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
3	2431	2
15	4123	4
3	3241	3
15	2134	6
3	4321	1

REALIZATION NUMBER 14

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
8	4213	2
15	2341	3
8	4132	4
15	1243	6
8	4321	1

REALIZATION NUMBER 15

SHIFT	ORDER	PARTITION
	4321	
16	3214	3
3	4132	4
16	1324	5
3	4213	2
16	2134	6
3	4321	1

REALIZATION NUMBER 16

SHIFT	ORDER	PARTITION
	4321	
16	3214	3
8	1423	4
16	4231	2
8	3124	5
16	1243	6
8	4321	1

REALIZATION NUMBER 17

SHIFT	ORDER	PARTITION
	4321	
8	2134	6
7	4213	2
8	1324	5
7	4132	4
8	3214	3
7	4321	1

REALIZATION NUMBER 18

SHIFT	ORDER	PARTITION
	4321	
10	2314	3
6	4213	2
10	1234	6
6	4132	4
10	3124	5
6	4321	1

Figure 17. Program for -/2/2(6) Charts

```
TREE: PROC OPTIONS (MAIN);

          /* NOTE: IN THIS PROGRAM A SHIFT HAS BEEN CALLED A GATE */
                /* OTHER MNEMONICS ARE FAIRLY OBVIOUS */
          /* OUTPUT WILL BE PUNCHED */

DCL (VAR(5), VARORD(5), I(5), II(5), COST1, COMP, CK, D,
   SEP, NAV, NO, GA, TGT, TPCOV, TCORD, TUORD(5),COST) FIXED BIN;
DCL GATE(19) LABEL INITIAL (CG1,CG2,CG3,CG4,CG5,CG6,CG7,CG8,CG9,
   CG10,CG11,CG12,CG13,CG14,CG15,CG16,CG17,CG18,CG19);
DCL CP ENTRY ((*) FIXED BIN,FIXED BIN) RETURNS (FIXED BIN);
DCL GG ENTRY ((*) FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,
   FIXED BIN);
DCL (PART(12,6 ), PCOV(1000,0:12), GT(1000,0:12),
     CORD(1000,0:12), UORD(1000,5)) FIXED BIN;

          /* COMPACT TO ONE WORD */
CP: PROC (VARORD,N) RETURNS(FIXED BIN);
DCL (VARORD(5),COMP) FIXED BIN;
   COMP=VARORD(1);
DO KK=2 TO N;
COMP=(10**(KK-1))*VARORD(KK)+COMP;
   END;
   RETURN (COMP);
   END CP;

          /* SIMULATION GATE */
GG: PROC (VARORD,IE,ID,IC,IB,IA);
DCL (VARORD(5), VAR(5)) FIXED BINARY;
VAR=VARORD;
VARORD(1)=VAR(IA);
VARORD(2)=VAR(IB);
VARORD(3)=VAR(IC);
VARORD(4)=VAR(ID);
VARORD(5)=VAR(IE);
END GG;

          /* CONTROL CARD  */
N=4; NAV=1000; GA=19; D=12; SEP=2; COST1=1; MAXCOST=4; /* 1/2/1 */

          /* GENERATE PARTITIONS */
          /* INITIALIZE */
VARORD=0;
DO I(1)=1 TO N;
VAR(I(1))=I(1);
END;
PUT FILE(SYSPNCH) EDIT ('1/2/1 (12)') (SKIP,A(10));
K=0;      /* K COUNTS NUMBER OF PARTITIONS */
```

Figure 18. Program for 1/2/1(12) Charts

```
          /* PICK ORDER FOR KTH PARTITION */
L121A: DO I(1)=1 TO N;
L121B: DO I(2)=1 TO N;
IF I(2)=I(1) THEN GO TO EL121B;
L121C: DO I(3)=1 TO N;
IF (I(3)=I(1))|(I(3)=I(2)) THEN GO TO EL121C;
L121D: DO I(4)=I(3)+1 TO N;
IF (I(4)=I(1))|(I(4)=I(2)) THEN GO TO EL121D;
K=K+1;
M=0;      /* M COUNTS SIZE OF EACH PARTITION */
          /* FORM ALL PERMUTATIONS OF ORIGINAL ORDER
            TO CREATE PARTITION */
PUT FILE(SYSPNCH) EDIT('PARTITION NUMBER',K) (SKIP,A(16),F(3));
VARORD(1)=VAR(I(1));
VARORD(4)=VAR(I(2));
L121CC: DO II(3)=3,4;
L121DD: DO II(4)=3,4;
IF II(3)=II(4) THEN GO TO EL121DD;
VARORD(2)=VAR(I(II(3)));
VARORD(3)=VAR(I(II(4)));
M=M+1;
          /* COMPACT, STORE, AND PRINT */
PART(K,M)=CP(VARORD,N);
PUT FILE(SYSPNCH) EDIT (PART(K,M)) (SKIP,F(5));
EL121DD: END L121DD;
EL121CC: END L121CC;
EL121D : END L121D ;
EL121C : END L121C ;
EL121B : END L121B ;
EL121A : END L121A ;

          /* GENERATE TREE TO INVESTIGATE GATES */
          /* INITIALIZE */
GT=-1; PCOV=0; TUORD=0; UORD=0; VAR=0;
DO K=1 TO N;
UORD(1,K)=K; TUORD(K)=K;
END;
L=0; NO=1;
CORD(NO,L)=CP(TUORD,N);
          /* 0TH ELEMENT OF GT WILL BE USED AS A FLAG TO TELL WHAT
             LEVEL THE ARRAY IS ON */
GT(NO,0)=0;
```

Figure 18. Program for 1/2/1(12) Charts

```
        /* MOVE TO NEXT LEVEL ON TREE. L CONTAINS CURRENT LEVEL */
  NLEV: M=0;
DO K=1 TO NAV;
IF GT(K,0)=L THEN M=M+1;
END;
PUT FILE(SYSPNCH) EDIT('LEVEL',L) (SKIP,A(5),F(3));
PUT FILE(SYSPNCH) EDIT ('NUMBER OF NODES ON TREE ARE',M)
(SKIP,A(27),F(5));
IF M>0 THEN GO TO INCRL;
PUT FILE(SYSPNCH) EDIT ('NO REALIZATION IS POSSIBLE')
(SKIP,A(26));
GO TO ETR;
  INCRL: L=L+1;
IF L>D THEN DO; L=D; GO TO PRNT; END;
NO=0;
          /* FIND NEXT AVAILABLE ARRAY TO BE INVESTIGATED. NO CONTAIN
             THE NUMBER OF THE ARRAY CURRENTLY UNDER INVESTIGATION */
  NXAV: NO=NO+1;
IF NO>NAV THEN GO TO NLEV;
IF GT(NO,0)¬=L-1 THEN GO TO NXAV;
DO K=1 TO N;
VAR(K)=UORD(NO,K);
END;
TGT=0;
          /* SELECT A TEMPORARY GATE TO BE CALLED.  TGT CONTAINS THE
             SUBSCRIPT OF THE GATE TO BE CALLED. COST CONTAINS THE
             COST OF CALLING THE GATE.  THE TOTAL COST SO FAR IS
             STORED AS A FLAG IN THE 0TH ELEMENT OF PCOV */
  CGT: TGT=TGT+1;
IF TGT>GA THEN DO; GT(NO,0)=-1; GO TO NXAV; END;
TUORD=VAR;
COST=0;
DO K=1 TO L-1;
IF GT(NO,K)=TGT THEN GO TO GATE(TGT);
END;
IF TGT<=COST1 THEN COST=1; ELSE COST=2;
IF PCOV(NO,0)+COST>MAXCOST THEN GO TO CGT;
GO TO GATE(TGT);
          /* ONLY GATES 3 7 8 12 15 16 */
CG1 : GO TO CGT;
CG2 : GO TO CGT;
CG3 : CALL GG( TUORD,5,1,2,4,3);   GO TO FPCOV;
CG4 : GO TO CGT;
CG5 : GO TO CGT;
CG6 : GO TO CGT;
CG7 : CALL GG( TUORD,5,1,4,3,2);   GO TO FPCOV;
CG8 : CALL GG( TUORD,5,2,1,3,4);   GO TO FPCOV;
CG9 : GO TO CGT;
```

Figure 18. Program for 1/2/1(12) Charts

```
CG10: GO TO CGT;
CG11: GO TO CGT;
CG12: CALL GG( TUORD,5,2,4,1,3);   GO TO FPCOV;
CG13: GO TO CGT;
CG14: GO TO CGT;
CG15: CALL GG( TUORD,5,3,1,4,2);   GO TO FPCOV;
CG16: CALL GG( TUORD,5,3,2,1,4);   GO TO FPCOV;
CG17: GO TO CGT;
CG18: GO TO CGT;
CG19: GO TO CGT;
         /* FIND PARTITION COVERED BY NEW VARIABLE ORDER */
  FPCOV: TCORD=CP(TUORD,N);
LOOP: DO K=1 TO D;
DO M=1 TO SEP;
IF TCORD=PART(K,M) THEN DO;
TPCOV=K;
GO TO CKPCOV; END;
END LOOP;
         /* CHECK PARTITION COVERED FOR DUPLICATION */
  CKPCOV: DO K=1 TO L-1;
IF TPCOV=PCOV(NO,K) THEN GO TO CGT;
END CKPCOV;
         /* FIND A VACANT ARRAY.  K CONTAINS THE SUBSCRIPT OF THE
            VACANT ARRAY FOUND */
  FVAC: DO K=1 TO NAV;
IF GT(K,0)=-1 THEN GO TO ADD;
END FVAC;
PUT FILE(SYSPNCH) EDIT ('NEED MORE ROOM LEVEL',L)
(SKIP,A(20),F(3));
PUT FILE(SYSPNCH) EDIT ('PROCESSING NUMBER',NO)
(SKIP,A(17),F(3));
GO TO ETR;
         /* TRANSFER OLD ARRAY INTO VACANT ARRAY AND ADD NEXT LEVEL*/
  ADD: DO M=0 TO L-1;
GT(K,M)=GT(NO,M);
PCOV(K,M)=PCOV(NO,M);
CORD(K,M)=CORD(NO,M);
END;
GT(K,0)=L;
PCOV(K,0)=PCOV(NO,0)+COST;
DO M=1 TO N;
UORD(K,M)=TUORD(M);
END;
GT(K,L)=TGT;
PCOV(K,L)=TPCOV;
CORD(K,L)=TCORD;
GO TO CGT;
```

Figure 18. Program for 1/2/1(12) Charts

```
        /* PRINT OUT ALL FINAL NODES WHICH RESTORE TO
           ORIGINAL ORDER */
  PRNT: DO COST=2 TO MAXCOST;
NUM1=-1; NUM2=0; NN=1;
PUT FILE(SYSPNCH) EDIT ('COST=',COST) (SKIP,A(5),F(3));
POUT: DO K=1 TO NAV;
IF GT(K,0)<L   THEN GO TO EPOUT;
IF PCOV(K,0)¬=COST THEN GO TO EPOUT;
IF CORD(K,D)¬=CORD(K,0) THEN GO TO EPOUT;
IF NN=1 THEN DO;
NUM1=NUM1+2;
NN=2;  K1=K;
GO TO EPOUT;  END;
NUM2=NUM2+2;
NN=1;  K2=K;
PUT FILE(SYSPNCH) EDIT ('REALIZATION NUMBER',NUM1,
'REALIZATION NUMBER',NUM2) (SKIP,A(18),F(3),X(14),A(18),F(3));
PUT FILE(SYSPNCH) EDIT ('SHIFT','ORDER','PARTITION','SHIFT',
'ORDER','PARTITION') (SKIP,A( 5),X(2),A(5),X(2),A(9),X(12),
A(5),X( 2),A(5),X(2),A(9));
PUT FILE(SYSPNCH) EDIT(CORD(K1,0),CORD(K2,0))
(SKIP,X(4),F(8),X(27),F(8));
DO M=1 TO D;
PUT FILE(SYSPNCH) EDIT (GT(K1,M),CORD(K1,M),PCOV(K1,M),
GT(K2,M),CORD(K2,M),PCOV(K2,M))
(SKIP,F(4),F(8),F(7),X(16),F(4),F(8),F(7));
END;
EPOUT: END POUT;
IF NUM1=-1 THEN DO;
PUT FILE(SYSPNCH) EDIT ('NO REALIZATIONS') (SKIP,A(15)); END;
IF NN=2 THEN DO;
PUT FILE(SYSPNCH) EDIT ('REALIZATION NUMBER',NUM1)
(SKIP,A(18),F(3));
PUT FILE(SYSPNCH) EDIT ('SHIFT','ORDER','PARTITION')
(SKIP,A(5),X(2),A(5),X(2),A(9));
PUT FILE(SYSPNCH) EDIT(CORD(K1,0)) (SKIP,X(4),F(8));
DO M=1 TO D;
PUT FILE(SYSPNCH) EDIT (GT(K1,M),CORD(K1,M),PCOV(K1,M))
(SKIP,F(4),F(8),F(7));
END; END;
END PRNT;
ETR: END TREE;
```

Figure 18. Program for 1/2/1(12) Charts

```
1/2/1 (12)


PARTITION NUMBER  1
 2431
 2341

PARTITION NUMBER  2
 3421
 3241

PARTITION NUMBER  3
 4321
 4231

PARTITION NUMBER  4
 1432
 1342

PARTITION NUMBER  5
 3412
 3142

PARTITION NUMBER  6
 4312
 4132

PARTITION NUMBER  7
 1423
 1243

PARTITION NUMBER  8
 2413
 2143

PARTITION NUMBER  9
 4213
 4123

PARTITION NUMBER 10
 1324
 1234

PARTITION NUMBER 11
 2314
 2134

PARTITION NUMBER 12
 3214
 3124
```

Figure 18. Program for 1/2/1(12) Charts

```
LEVEL  0
NUMBER OF NODES ON TREE ARE     1

LEVEL  1
NUMBER OF NODES ON TREE ARE     6

LEVEL  2
NUMBER OF NODES ON TREE ARE    36

LEVEL  3
NUMBER OF NODES ON TREE ARE    84

LEVEL  4
NUMBER OF NODES ON TREE ARE   156

LEVEL  5
NUMBER OF NODES ON TREE ARE   248

LEVEL  6
NUMBER OF NODES ON TREE ARE   360

LEVEL  7
NUMBER OF NODES ON TREE ARE   424

LEVEL  8
NUMBER OF NODES ON TREE ARE   408

LEVEL  9
NUMBER OF NODES ON TREE ARE   352

LEVEL 10
NUMBER OF NODES ON TREE ARE   232

LEVEL 11
NUMBER OF NODES ON TREE ARE   128

LEVEL 12
NUMBER OF NODES ON TREE ARE    80
```

Figure 18. Program for 1/2/1(12) Charts

COST= 2
NO REALIZATIONS

COST= 3
NO REALIZATIONS

COST= 4

REALIZATION NUMBER 1

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
16	4132	6
12	3421	2
12	2314	11
12	1243	7
16	2431	1
12	3214	12
12	1342	4
12	4123	9
16	1234	10
12	3142	5
12	4321	3

REALIZATION NUMBER 2

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
16	1423	7
15	4312	6
15	3241	2
15	2134	11
16	1342	4
15	3214	12
15	2431	1
15	4123	9
16	1234	10
15	2413	8
15	4321	3

REALIZATION NUMBER 3

SHIFT	ORDER	PARTITION
	4321	
16	3214	12
12	1342	4
12	4123	9
12	2431	1
16	4312	6
12	1423	7
12	2134	11
12	3241	2
16	2413	8
12	1234	10
12	3142	5
12	4321	3

REALIZATION NUMBER 4

SHIFT	ORDER	PARTITION
	4321	
16	3214	12
15	2431	1
15	4123	9
15	1342	4
16	3421	2
15	4132	6
15	1243	7
15	2314	11
16	3142	5
15	1234	10
15	2413	8
15	4321	3

Figure 18. Program for 1/2/1(12) Charts

REALIZATION NUMBER 5

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
12	1234	10
12	3142	5
3	2431	1
12	3214	12
12	1342	4
12	4123	9
3	3241	2
12	4312	6
12	1423	7
12	2134	11
3	4321	3

REALIZATION NUMBER 6

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
12	1234	10
16	2341	1
12	4213	9
12	1432	4
12	3124	12
16	1243	7
12	4132	6
12	3421	2
12	2314	11
16	3142	5
12	4321	3

REALIZATION NUMBER 7

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
15	1234	10
16	2341	1
15	3124	12
15	1432	4
15	4213	9
16	2134	11
15	1423	7
15	4312	6
15	3241	2
16	2413	8
15	4321	3

REALIZATION NUMBER 8

SHIFT	ORDER	PARTITION
	4321	
3	1243	7
12	4132	6
12	3421	2
12	2314	11
3	4123	9
12	2431	1
12	3214	12
12	1342	4
3	2413	8
12	1234	10
12	3142	5
12	4321	3

REALIZATION NUMBER 9

SHIFT	ORDER	PARTITION
	4321	
3	1243	7
15	2314	11
15	3421	2
15	4132	6
3	2341	1
15	3124	12
15	1432	4
15	4213	9
3	3142	5
15	1234	10
15	2413	8
15	4321	3

REALIZATION NUMBER 10

SHIFT	ORDER	PARTITION
	4321	
7	1432	4
12	3124	12
12	2341	1
12	4213	9
7	3421	2
12	2314	11
12	1243	7
12	4132	6
7	2413	8
12	1234	10
12	3142	5
12	4321	3

Figure 18. Program for 1/2/1(12) Charts

REALIZATION NUMBER 11

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
12	1234	10
12	3142	5
16	1423	7
12	2134	11
12	3241	2
12	4312	6
16	3124	12
12	2341	1
12	4213	9
12	1432	4
16	4321	3

REALIZATION NUMBER 12

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
15	1234	10
15	2413	8
16	4132	6
15	1243	7
15	2314	11
15	3421	2
16	4213	9
15	2341	1
15	3124	12
15	1432	4
16	4321	3

REALIZATION NUMBER 13

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
3	3124	12
12	2341	1
12	4213	9
12	1432	4
3	2314	11
12	1243	7
12	4132	6
12	3421	2
3	1234	10
12	3142	5
12	4321	3

REALIZATION NUMBER 14

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
3	2431	1
15	4123	9
15	1342	4
15	3214	12
3	4132	6
15	1243	7
15	2314	11
15	3421	2
3	1234	10
15	2413	8
15	4321	3

REALIZATION NUMBER 15

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
7	3241	2
12	4312	6
12	1423	7
12	2134	11
7	4213	9
12	1432	4
12	3124	12
12	2341	1
7	1234	10
12	3142	5
12	4321	3

REALIZATION NUMBER 16

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
8	1342	4
12	4123	9
12	2431	1
12	3214	12
8	1423	7
12	2134	11
12	3241	2
12	4312	6
8	1234	10
12	3142	5
12	4321	3

Figure 18. Program for 1/2/1(12) Charts

REALIZATION NUMBER 17

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
12	1234	10
7	4123	9
12	2431	1
12	3214	12
12	1342	4
7	2134	11
12	3241	2
12	4312	6
12	1423	7
7	3142	5
12	4321	3

REALIZATION NUMBER 18

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
7	2314	11
15	3421	2
15	4132	6
15	1243	7
7	3124	12
15	1432	4
15	4213	9
15	2341	1
7	1234	10
15	2413	8
15	4321	3

REALIZATION NUMBER 19

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
8	4213	9
15	2341	1
15	3124	12
15	1432	4
8	3241	2
15	2134	11
15	1423	7
15	4312	6
8	1234	10
15	2413	8
15	4321	3

REALIZATION NUMBER 20

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
12	1234	10
12	3142	5
8	4213	9
12	1432	4
12	3124	12
12	2341	1
8	4132	6
12	3421	2
12	2314	11
12	1243	7
8	4321	3

REALIZATION NUMBER 21

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
15	1234	10
15	2413	8
7	3241	2
15	2134	11
15	1423	7
15	4312	6
7	2431	1
15	4123	9
15	1342	4
15	3214	12
7	4321	3

REALIZATION NUMBER 22

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
12	1234	10
8	3421	2
12	2314	11
12	1243	7
12	4132	6
8	3214	12
12	1342	4
12	4123	9
12	2431	1
8	3142	5
12	4321	3

Figure 18. Program for 1/2/1(12) Charts

REALIZATION NUMBER 23

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
15	1234	10
7	4123	9
15	1342	4
15	3214	12
15	2431	1
7	1243	7
15	2314	11
15	3421	2
15	4132	6
7	2413	8
15	4321	3

REALIZATION NUMBER 24

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
15	1234	10
15	2413	8
3	3124	12
15	1432	4
15	4213	9
15	2341	1
3	1423	7
15	4312	6
15	3241	2
15	2134	11
3	4321	3

REALIZATION NUMBER 25

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
15	1234	10
3	4312	6
15	3241	2
15	2134	11
15	1423	7
3	3214	12
15	2431	1
15	4123	9
15	1342	4
3	2413	8
15	4321	3

REALIZATION NUMBER 26

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
12	1234	10
12	3142	5
7	2314	11
12	1243	7
12	4132	6
12	3421	2
7	1342	4
12	4123	9
12	2431	1
12	3214	12
7	4321	3

REALIZATION NUMBER 27

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
15	1234	10
8	3421	2
15	4132	6
15	1243	7
15	2314	11
8	1432	4
15	4213	9
15	2341	1
15	3124	12
8	2413	8
15	4321	3

REALIZATION NUMBER 28

SHIFT	ORDER	PARTITION
	4321	
15	3142	5
15	1234	10
15	2413	8
8	1342	4
15	3214	12
15	2431	1
15	4123	9
8	2314	11
15	3421	2
15	4132	6
15	1243	7
8	4321	3

Figure 18. Program for 1/2/1(12) Charts

REALIZATION NUMBER 29

SHIFT	ORDER	PARTITION
	4321	
7	1432	4
15	4213	9
15	2341	1
15	3124	12
7	4312	6
15	3241	2
15	2134	11
15	1423	7
7	3142	5
15	1234	10
15	2413	8
15	4321	3

REALIZATION NUMBER 30

SHIFT	ORDER	PARTITION
	4321	
8	2134	11
12	3241	2
12	4312	6
12	1423	7
8	2341	1
12	4213	9
12	1432	4
12	3124	12
8	2413	8
12	1234	10
12	3142	5
12	4321	3

REALIZATION NUMBER 31

SHIFT	ORDER	PARTITION
	4321	
8	2134	11
15	1423	7
15	4312	6
15	3241	2
8	4123	9
15	1342	4
15	3214	12
15	2431	1
8	3142	5
15	1234	10
15	2413	8
15	4321	3

REALIZATION NUMBER 32

SHIFT	ORDER	PARTITION
	4321	
12	2413	8
12	1234	10
3	4312	6
12	1423	7
12	2134	11
12	3241	2
3	1432	4
12	3124	12
12	2341	1
12	4213	9
3	3142	5
12	4321	3

Figure 18. Program for 1/2/1(12) Charts

APPENDIX B

COMPUTER PROGRAM FOR DERIVING MINTERM ORDERS FROM VARIABLE ORDERS

The computer program shown here uses the weighted conversion technique discussed in Section 3.2 for obtaining the required minterm order of a decomposition chart when its variable order is given. A starting minterm order, listing the minterms from 0 to 15, is assumed. This corresponds to the variable order 4321. The shifts in the cost 3,-/3/1(4) library from Appendix A are applied to this starting variable order and the implied minterm orders are found.

```
MINTRMS: PROC OPTIONS(MAIN);

         /* NOTE: SOME CARDS FROM THE OTHER PROGRAM HAVE BEEN USED
          MNEMONICS ARE NOT ALWAYS OBVIOUS */
         /* OUTPUT WILL BE PUNCHED */

DCL (TUORD(5), VAR(5), GA, MINTR1(0:31),G1, CORD1,
  MINTR2(0:31), G2, CORD2, CVAR) FIXED BIN;
DCL GATE(19) LABEL INITIAL (CG1,CG2,CG3,CG4,CG5,CG6,CG7,CG8,CG9,
   CG10,CG11,CG12,CG13,CG14,CG15,CG16,CG17,CG18,CG19);
DCL CP ENTRY ((*) FIXED BIN,FIXED BIN) RETURNS (FIXED BIN);
DCL GG ENTRY ((*) FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN
   FIXED BIN);
DCL MINORD ENTRY ( (*) FIXED BIN, (*) FIXED BIN, FIXED BIN);

         /* COMPACT TO ONE WORD */
CP: PROC (VARORD,N) RETURNS(FIXED BIN);
DCL (VARORD(5),COMP) FIXED BIN;
   COMP=VARORD(1);
DO KK=2 TO N;
COMP=(10**(KK-1))*VARORD(KK)+COMP;
   END;
   RETURN (COMP);
   END CP;

         /* SIMULATION GATE */
GG: PROC (VARORD,IE,ID,IC,IB,IA);
DCL (VARORD(5), VAR(5)) FIXED BINARY;
VAR=VARORD;
VARORD(1)=VAR(IA);
VARORD(2)=VAR(IB);
VARORD(3)=VAR(IC);
VARORD(4)=VAR(ID);
VARORD(5)=VAR(IE);
END GG;

         /* WEIGHTED CONVERSION TO BASE TEN */
    MINORD: PROC (VARORD,MINTR,N);
DCL (MINTR(0:31),VARORD(5), WT(5) ) FIXED BIN;
     L: DO I=1 TO N;
     WT(I)=2**(VARORD(I)-1);
     END L;
     MINTR=0;
     LOOPS: DO K=1 TO N;
      DO I=2**(K-1) BY 2**K TO (2**N)-1;
       DO J=0 TO (2**(K-1))-1;
     MINTR(I+J)=MINTR(I+J)+WT(K);
     END LOOPS;
     END MINORD;
```

Figure 19. Program to find minterm orders for cost 3, -/3/1(4) library

```
        /* CONTROL CARD */
N=4; GA=19;

        /* FIND VARIABLE AND MINTERM SHIFTS */
        /* INITIALIZE */
VAR=0;
DO I=1 BY 1 TO N;
VAR(I)=I;
END;
CVAR=CP(VAR,N);
M=1;
        /* SELECT GATE */
  SGT: DO L=1 TO GA;
TUORD=VAR;
GO TO GATE(L);
        /* ONLY GATES 1 3 6 7 8 10 12 15 16 */
CG1 : CALL GG( TUORD,5,4,2,1,3);   GO TO FPCOV;
CG2 : GO TO CGT;
CG3 : CALL GG( TUORD,5,1,2,4,3);   GO TO FPCOV;
CG4 : GO TO CGT;
CG5 : GO TO CGT;
CG6 : CALL GG( TUORD,5,1,4,2,3);   GO TO FPCOV;
CG7 : CALL GG( TUORD,5,1,4,3,2);   GO TO FPCOV;
CG8 : CALL GG( TUORD,5,2,1,3,4);   GO TO FPCOV;
CG9 : GO TO CGT;
CG10: CALL GG( TUORD,5,2,3,1,4);   GO TO FPCOV;
CG11: GO TO CGT;
CG12: CALL GG( TUORD,5,2,4,1,3);   GO TO FPCOV;
CG13: GO TO CGT;
CG14: GO TO CGT;
CG15: CALL GG( TUORD,5,3,1,4,2);   GO TO FPCOV;
CG16: CALL GG( TUORD,5,3,2,1,4);   GO TO FPCOV;
CG17: GO TO CGT;
CG18: GO TO CGT;
CG19: GO TO CGT;
        /* SAVE TWO GATES TO BE PRINTED AT ONE TIME */
FPCOV: IF M=1 THEN DO;
G1=L;
CORD1=CP(TUORD,N);
CALL MINORD(TUORD,MINTR1,N);
M=2;
GO TO CGT; END;
G2=L;
CORD2=CP(TUORD,N);
CALL MINORD(TUORD,MINTR2,N);
M=1;
```

Figure 19. Program to find minterm orders for cost 3, -/3/1(4) library

```
          /* PRINT OUT RESULTS */
PUT FILE(SYSPNCH ) EDIT('GATE',G1,'GATE',G2)
(SKIP,A(4),F(3),X(28),A(4),F(3));
PUT FILE(SYSPNCH ) EDIT('CHANGE IN VARIABLE ORDER',
'CHANGE IN VARIABLE ORDER') (SKIP,A(24),X(11),A(24));
PUT FILE(SYSPNCH ) EDIT(CVAR,CORD1,CVAR,CORD2)
(SKIP,F(5),X(14),F(5),X(11),F(5),X(14),F(5));
PUT FILE(SYSPNCH ) EDIT ('CHANGE IN MINTERM ORDER',
'CHANGE IN MINTERM ORDER') (SKIP,A(23),X(12),A(23));
DO J=0 TO (2**N)-1;
PUT FILE(SYSPNCH ) EDIT (J ,MINTR1(J ),J ,MINTR2(J ))
(SKIP,F(5),X(10),F(5),X(15),F(5),X(10),F(5));
END;
CGT: END SGT;
IF M=2 THEN DO;
PUT FILE(SYSPNCH ) EDIT('GATE',G1)
(SKIP,A(4),F(3));
PUT FILE(SYSPNCH ) EDIT('CHANGE IN VARIABLE ORDER')
(SKIP,A(24));
PUT FILE(SYSPNCH ) EDIT(CVAR,CORD1)
(SKIP,F(5),X(14),F(5));
PUT FILE(SYSPNCH ) EDIT ('CHANGE IN MINTERM ORDER')
(SKIP,A(23));
DO J=0 TO (2**N)-1;
PUT FILE(SYSPNCH ) EDIT (J ,MINTR1(J ))
(SKIP,F(5),X(10),F(5));
END; END;
END MINTRMS;
```

Figure 19. Program to find minterm orders for cost 3, -/3/1(4) library

GATE 1

CHANGE IN VARIABLE ORDER	
4321	4213
CHANGE IN MINTERM ORDER	
0	0
1	4
2	1
3	5
4	2
5	6
6	3
7	7
8	8
9	12
10	9
11	13
12	10
13	14
14	11
15	15

GATE 3

CHANGE IN VARIABLE ORDER	
4321	1243
CHANGE IN MINTERM ORDER	
0	0
1	4
2	8
3	12
4	2
5	6
6	10
7	14
8	1
9	5
10	9
11	13
12	3
13	7
14	11
15	15

GATE 6

CHANGE IN VARIABLE ORDER	
4321	1423
CHANGE IN MINTERM ORDER	
0	0
1	4
2	2
3	6
4	8
5	12
6	10
7	14
8	1
9	5
10	3
11	7
12	9
13	13
14	11
15	15

GATE 7

CHANGE IN VARIABLE ORDER	
4321	1432
CHANGE IN MINTERM ORDER	
0	0
1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	1
9	3
10	5
11	7
12	9
13	11
14	13
15	15

Figure 19. Program to find minterm orders for cost 3, -/3/1(4) library

GATE 8

CHANGE IN VARIABLE ORDER 4321	2134
CHANGE IN MINTERM ORDER	
0	0
1	8
2	4
3	12
4	1
5	9
6	5
7	13
8	2
9	10
10	6
11	14
12	3
13	11
14	7
15	15

GATE 10

CHANGE IN VARIABLE ORDER 4321	2314
CHANGE IN MINTERM ORDER	
0	0
1	8
2	1
3	9
4	4
5	12
6	5
7	13
8	2
9	10
10	3
11	11
12	6
13	14
14	7
15	15

GATE 12

CHANGE IN VARIABLE ORDER 4321	2413
CHANGE IN MINTERM ORDER	
0	0
1	4
2	1
3	5
4	8
5	12
6	9
7	13
8	2
9	6
10	3
11	7
12	10
13	14
14	11
15	15

GATE 15

CHANGE IN VARIABLE ORDER 4321	3142
CHANGE IN MINTERM ORDER	
0	0
1	2
2	8
3	10
4	1
5	3
6	9
7	11
8	4
9	6
10	12
11	14
12	5
13	7
14	13
15	15

Figure 19. Program to find minterm orders for cost 3, -/3/1(4) library

```
GATE 16

CHANGE IN VARIABLE ORDER
 4321                3214

CHANGE IN MINTERM ORDER
    0                0
    1                8
    2                1
    3                9
    4                2
    5               10
    6                3
    7               11
    8                4
    9               12
   10                5
   11               13
   12                6
   13               14
   14                7
   15               15
```

Figure 19. Program to find minterm orders for cost 3, -/3/1(4) library